Overview

- Finish timing (clock) recovery
  - Manchester, 4B/5B, etc.

- Methods to share physical media: multiple access
  - Fixed partitioning
  - Random access

- Channelizing mechanisms

- Contention-based mechanisms
  - Aloha
The Importance of Phase

- Need to determine when to START sampling, too
Clock Recovery

- Using a training sequence to get receiver lined up
  - Send a few, known initial training bits
  - Adds inefficiency: only \( m \) data bits out of \( n \) transmitted

- Need to combat clock drift as signal proceeds
  - Use transitions to keep clocks synched up

- Question is, how often do we do this?
  - Quick and dirty every time: asynchronous coding
  - Spend a lot of effort to get it right, but amortize over lots of data: synchronous coding
Asynchronous Coding

- Encode several bits (e.g. 7) together with a leading “start bit” and trailing “stop bit”
- Data can be sent at any time

- Start bit transition kicks off sampling intervals
  - Can only run for a short while before drifting
Example: RS232 serial lines

- Uses two voltage levels (+15V, -15V), to encode single bit binary symbols
- Needs long idle time – limited transmit rate
Synchronous coding

- Encode many bits (thousands) together
  - Amortize cost of learning clock information from start bits (preamble) and stop bits (trailer)
  - Continuously “learn” clock from data stream
    » Watch for 0-1 or 1-0 transitions, and adjust clock
    » Called clock recovery process

- Examples
  - NRZ
  - NRZI
  - Manchester
  - 4B/5B
  - Many others…
Synchronous Coding

- Asynchronous receiver phase locks each symbol
  - Takes time, limiting transmission rates

- So, start symbols need to be extra slow
  - Need to fire up the clock, which takes time

- Instead, let’s do this training once, then just keep sync
  - Need to continually adjust clock as signal arrives
  - Ever hear of Phase Lock Loops (PLLs)?

- Basic idea is to use transitions to lock in
Non-Return to Zero (NRZ)

- Signal to Data
  - High ⇒ 1
  - Low ⇒ 0

- Comments
  - Transitions maintain clock synchronization
  - Long strings of 0s confused with no signal
  - Long strings of 1s causes *baseline wander*
    - We use average signal level to infer high vs low
  - Both inhibit clock recovery

```
Bits: 0 0 1 0 1 1 1 1 0 1 0 0 0 0 1 0

NRZ:  
```
Non-Return to Zero Inverted (NRZI)

- **Signal to Data**
  - Transition $\Rightarrow 1$
  - Maintain $\Rightarrow 0$

- **Comments**
  - Solves series of 1s, but not 0s
Manchester Encoding
(10Mbps Ethernet)

- **Signal to Data**
  - XOR NRZ data with senders clock signal
  - High to low transition $\Rightarrow$ 1
  - Low to high transition $\Rightarrow$ 0

- **Comments**
  - Solves clock recovery problem
  - Only 50% efficient (½ bit per transition)
  - Still need preamble (typically 0101010101… trailing 11 in Ethernet)
Goal: address inefficiency of Manchester encoding, while avoiding long periods of low signals

Solution:
- Use five bits to encode every sequence of four bits
- No 5 bit code has more than one leading 0 and two trailing 0’s
- Use NRZI to encode the 5 bit codes
- Efficiency is 80%

<table>
<thead>
<tr>
<th>4-bit</th>
<th>5-bit</th>
<th>4-bit</th>
<th>5-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>11110</td>
<td>1000</td>
<td>10010</td>
</tr>
<tr>
<td>0001</td>
<td>01001</td>
<td>1001</td>
<td>10011</td>
</tr>
<tr>
<td>0010</td>
<td>10100</td>
<td>1010</td>
<td>10110</td>
</tr>
<tr>
<td>0011</td>
<td>10101</td>
<td>1011</td>
<td>10111</td>
</tr>
<tr>
<td>0100</td>
<td>01010</td>
<td>1100</td>
<td>11010</td>
</tr>
<tr>
<td>0101</td>
<td>01011</td>
<td>1101</td>
<td>11011</td>
</tr>
<tr>
<td>0110</td>
<td>01110</td>
<td>1110</td>
<td>11100</td>
</tr>
<tr>
<td>0111</td>
<td>01111</td>
<td>1111</td>
<td>11101</td>
</tr>
</tbody>
</table>
Encoding/Decoding Summary

- Signaling & Modulation
  - Transforming digital signal to and from analog representation
  - Fundamental limits (Shannon)
  - Lots of ways to encode signal (modulation) onto a given medium

- Clock recovery
  - Receiver needs to adjust its sampling times to best extract signal from channel
  - Sender can code signal to make it far easier to do this
Fixed Partitioning of Channels

- Need to share media with multiple nodes ($n$)
  - Multiple *simultaneous* conversations

- A simple solution
  - Divide the channel into multiple, separate channels

- Channels are physically separate
  - Bitrate of the link is split across channels
  - Nodes can only send/receive on their assigned channel

- Several different ways to do it
  - ____ Multiple Access madlibs…
Frequency Division (FDMA)

- Divide bandwidth of $f$ Hz into $n$ channels each with bandwidth $f/n$ Hz
  - Easy to implement, but unused subchannels go idle
  - Used by traditional analog cell phone service, radio, TV

CSE 123 – Lecture 27: Media Access Control
Time Division (TDMA)

- Divide channel into rounds of $n$ time slots each
  - Assign different hosts to different time slots within a round
  - Unused time slots are idle
  - Used in GSM cell phones & digital cordless phones

- Example with 1-second rounds
  - $n=4$ timeslots (250ms each) per round
Code Division (CDMA)

- Do nothing to physically separate the channels
  - All stations transmit at same time in same frequency bands
  - One of so-called spread-spectrum techniques

- Sender modulates their signal on top of unique code
  - Sort of like the way Manchester modulates on top of clock
  - The bit rate of resulting signal much lower than entire channel

- Receiver applies code filter to extract desired sender
  - All other senders seem like noise with respect to signal

- Used in older digital cellular technologies
Partitioning Visualization (the rainbow cake)

- **FDMA**
- **TDMA**
- **CDMA**

Courtesy Takashi Inoue
Problem w/Channel partitioning

- Not terribly well suited for random access usage
  - Why?

- Instead, design schemes for more common situations
  - Not all nodes want to send all the time
  - Don’t have a fixed number of nodes

- Potentially higher throughput for transmissions
  - Active nodes get full channel bandwidth
Aloha

- Designed in 1970 to support wireless data connectivity
  - Between Hawaiian Islands!

- Goal: distributed access control (no central arbitrator)
  - Over a shared broadcast channel

- Aloha protocol in a nutshell:
  - When you have data send it
  - If data doesn’t get through (receiver sends acknowledgement) then retransmit after a random delay
  - Why not a fixed delay?
Collisions

- Frame sent at $t_0$ collides with frames sent in $[t_0-1, t_0+1]$
  - Assuming unit-length frames
  - Ignores propagation delay
Slotted Aloha

- Time is divided into equal size slots (frame size)
- Host wanting to transmit starts at start of next slot
  - Retransmit like w/Aloha, but quantize to nearest next slot
- Requires **time synchronization** between hosts

![Diagram showing slotted Aloha with nodes 1, 2, and 3 transmitting]

Success (S), Collision (C), Empty (E) slots
Q: What is max fraction slots successful?
A: Suppose $n$ stations have packets to send
   - Each transmits in slot with probability $p$
   - $\text{Prob}\{\text{successful transmission}\}$, $S$, is:

   $$S = p(1-p)^{(n-1)}$$

   - any of $n$ nodes:

   $$S = \text{Prob}\{\text{one transmits}\} = np(1-p)^{(n-1)}$$

   (optimal $p$ as $n\to\infty = 1/n$)

   $$= \frac{1}{e} = .37$$

---

At best: channel used for useful transmissions 37% of time!
For Next Time

- Keep going on the project…