

Lecture 27:

Media Access Control

CSE 123: Computer Networks

Aaron Schulman



Overview

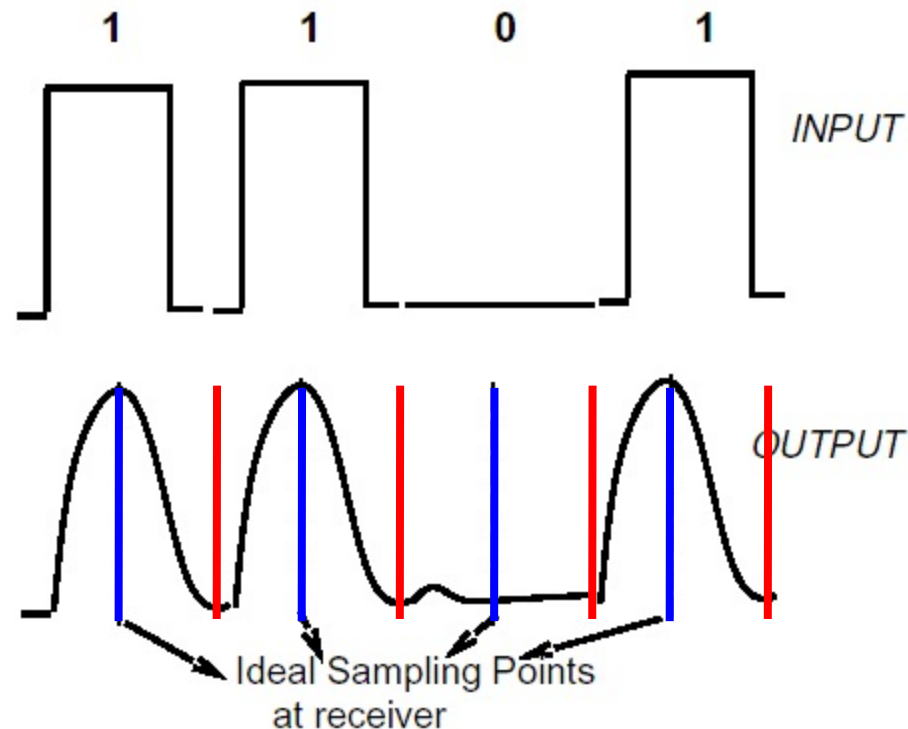


- Finish timing (clock) recovery
 - ◆ Manchester, 4B/5B, etc.
- Methods to share physical media: **multiple access**
 - ◆ Fixed partitioning
 - ◆ Random access
- Channelizing mechanisms
- Contention-based mechanisms
 - ◆ Aloha



The Importance of Phase

- Need to determine when to START sampling, too





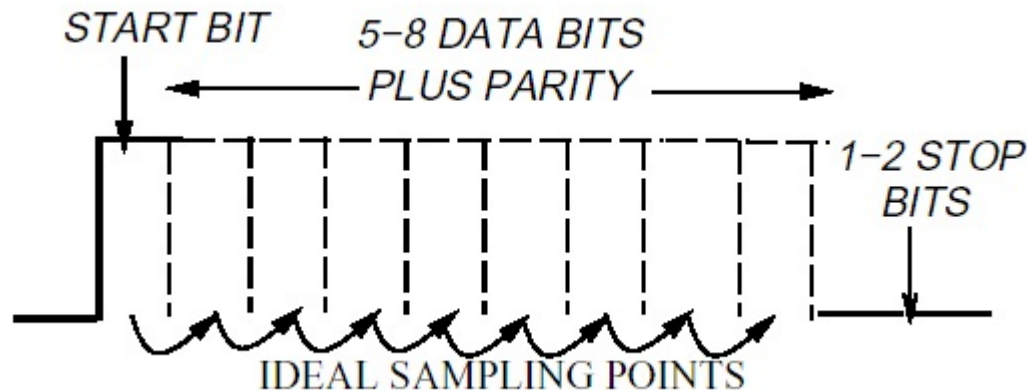
Clock Recovery

- 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

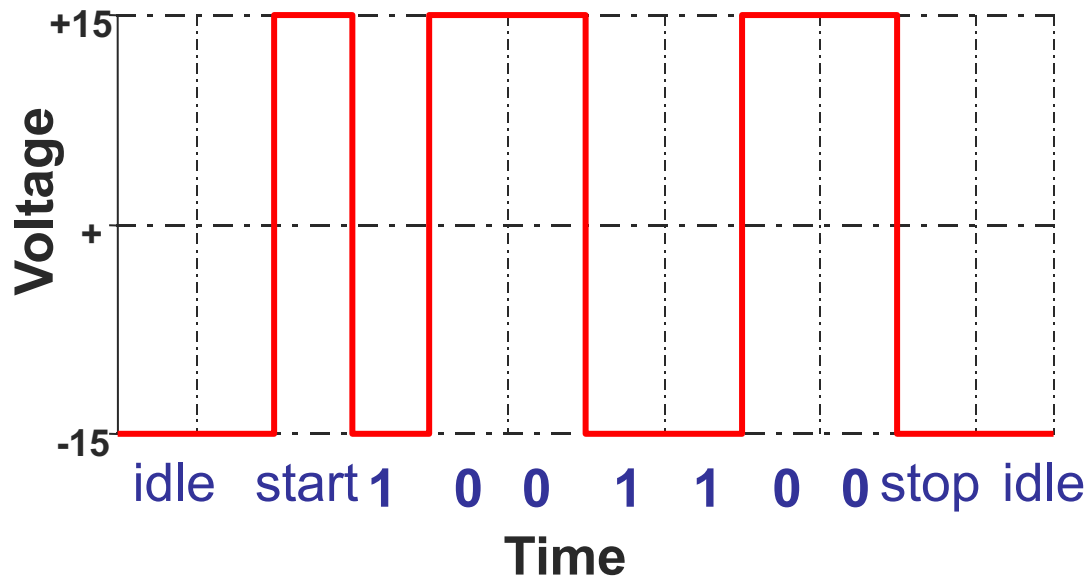


- Start bit transition kicks off sampling intervals
 - Can only run for a short while before drifting

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...



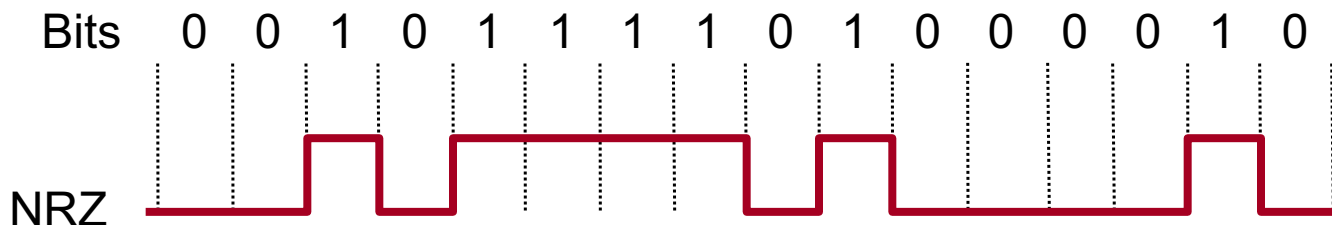
Synchronous Coding

- 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)

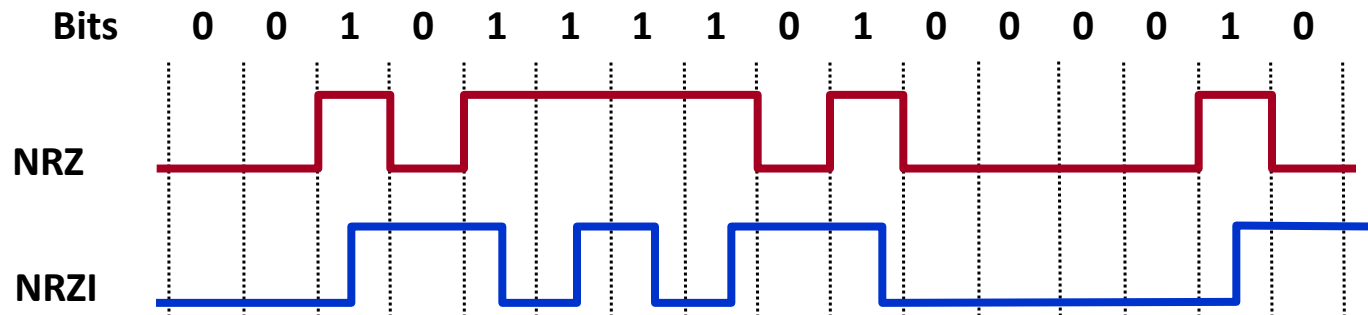
- Signal to Data
 - ◆ High $\Rightarrow$ 1
 - ◆ Low $\Rightarrow$ 0
- Comments
 - ◆ Transitions maintain clock synchronization
 - ◆ Long strings of 0s confused with no signal
 - ◆ Long strings of 1s causes *baseline wander*
 - » We use average signal level to infer high vs low
 - ◆ Both inhibit clock recovery



Non-Return to Zero Inverted (NRZI)



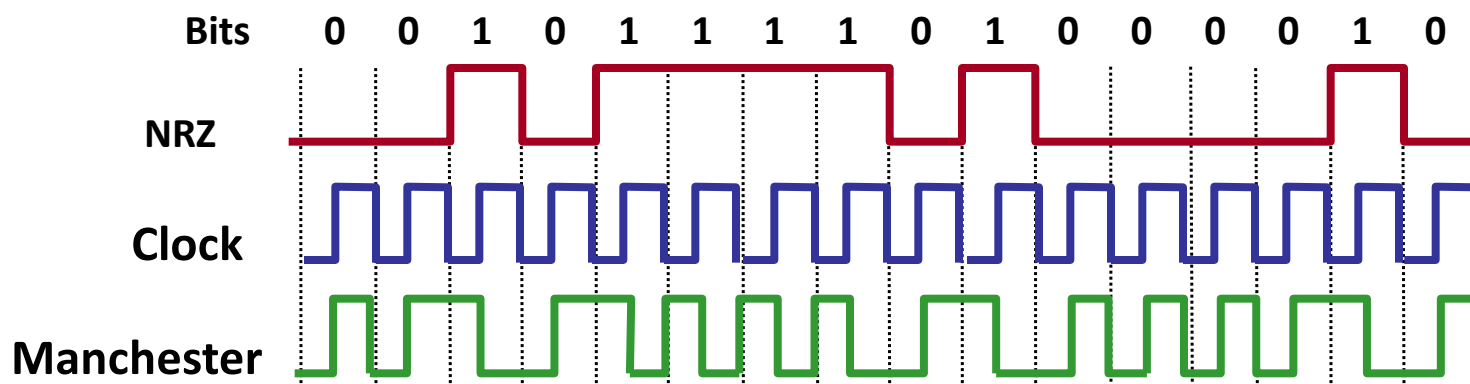
- Signal to Data
 - ◆ Transition $\Rightarrow$ 1
 - ◆ Maintain $\Rightarrow$ 0
- Comments
 - ◆ Solves series of 1s, but not 0s



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 ($\frac{1}{2}$ bit per transition)
 - ◆ Still need preamble (typically 0101010101... trailing 11 in Ethernet)



4B/5B (100Mbps 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%

4-bit	5-bit
0000	11110
0001	01001
0010	10100
0011	10101
0100	01010
0101	01011
0110	01110
0111	01111

4-bit	5-bit
1000	10010
1001	10011
1010	10110
1011	10111
1100	11010
1101	11011
1110	11100
1111	11101

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

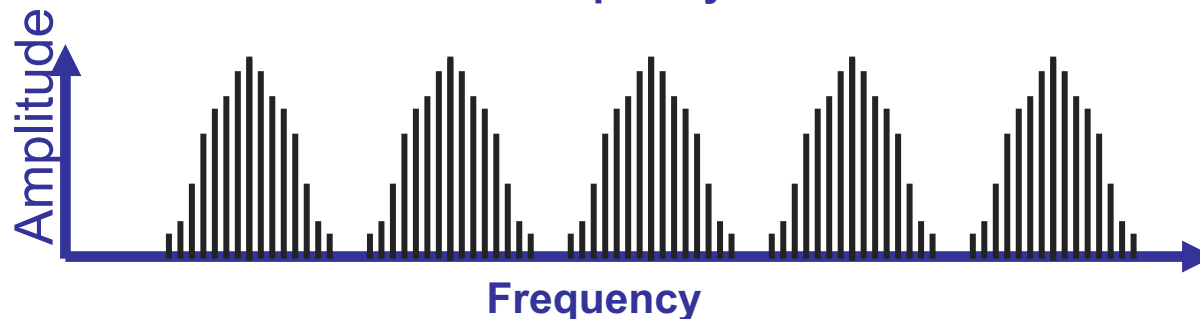
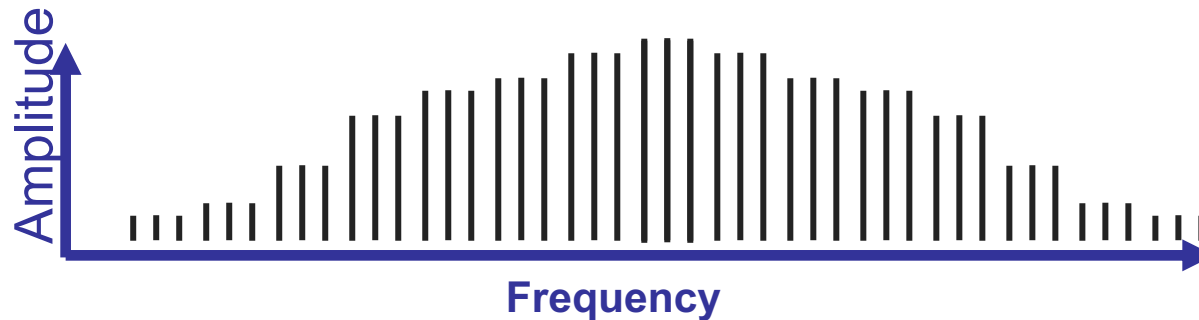


- Need to share media with multiple nodes (n)
 - ◆ Multiple *simultaneous* conversations
- A simple solution
 - ◆ Divide the channel into multiple, separate **channels**
- Channels are physically separate
 - ◆ Bitrate of the link is split across channels
 - ◆ Nodes can only send/receive on their assigned channel
- Several different ways to do it
 - ◆ _____ **Multiple Access** madlibs...

Frequency Division (FDMA)



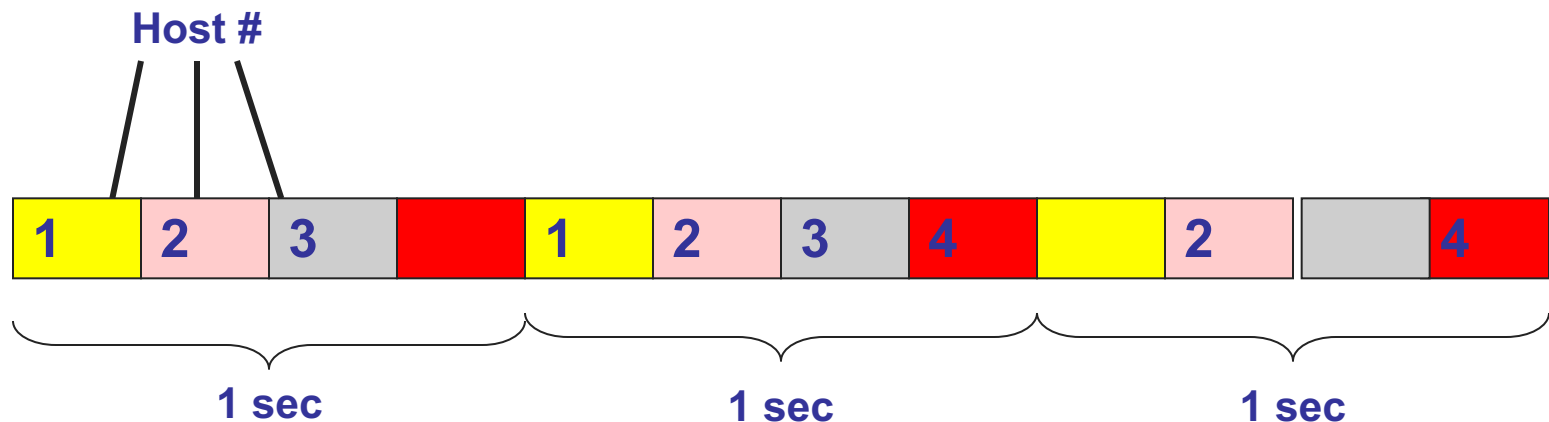
- Divide bandwidth of f Hz into n channels each with bandwidth f/n Hz
 - ◆ Easy to implement, but unused subchannels go idle
 - ◆ Used by traditional analog cell phone service, radio, TV





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

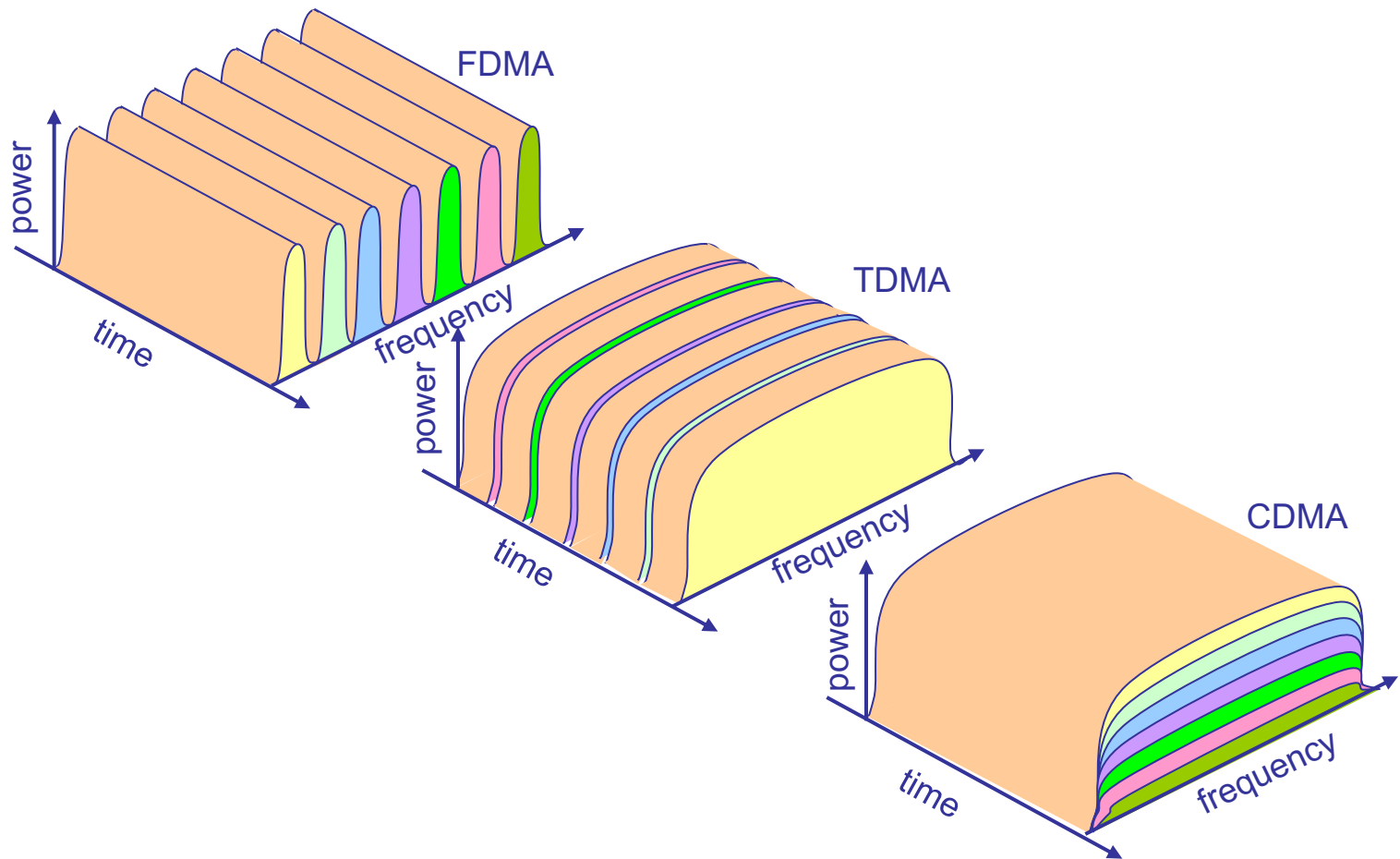


Code Division (CDMA)



- 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)



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

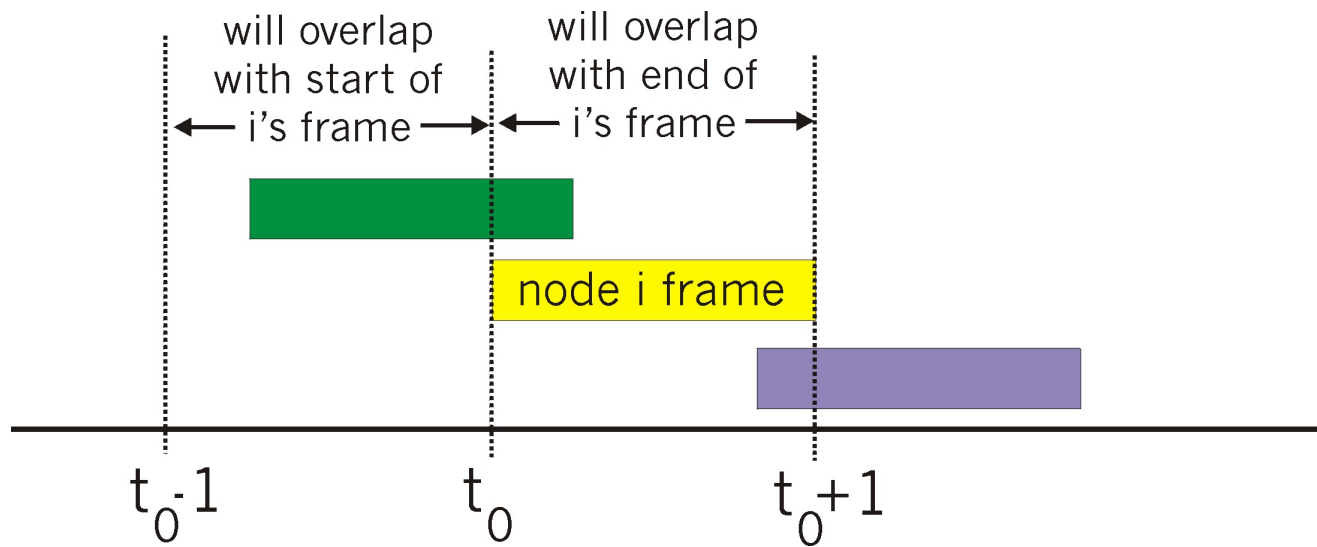


- Designed in 1970 to support wireless data connectivity
 - ◆ Between Hawaiian Islands!
- Goal: distributed access control (no central arbitrator)
 - ◆ Over a shared *broadcast* channel
- Aloha protocol in a nutshell:
 - ◆ When you have data **send it**
 - ◆ If data doesn't get through (receiver sends acknowledgement) then **retransmit after a random delay**
 - ◆ Why not a fixed delay?

Collisions



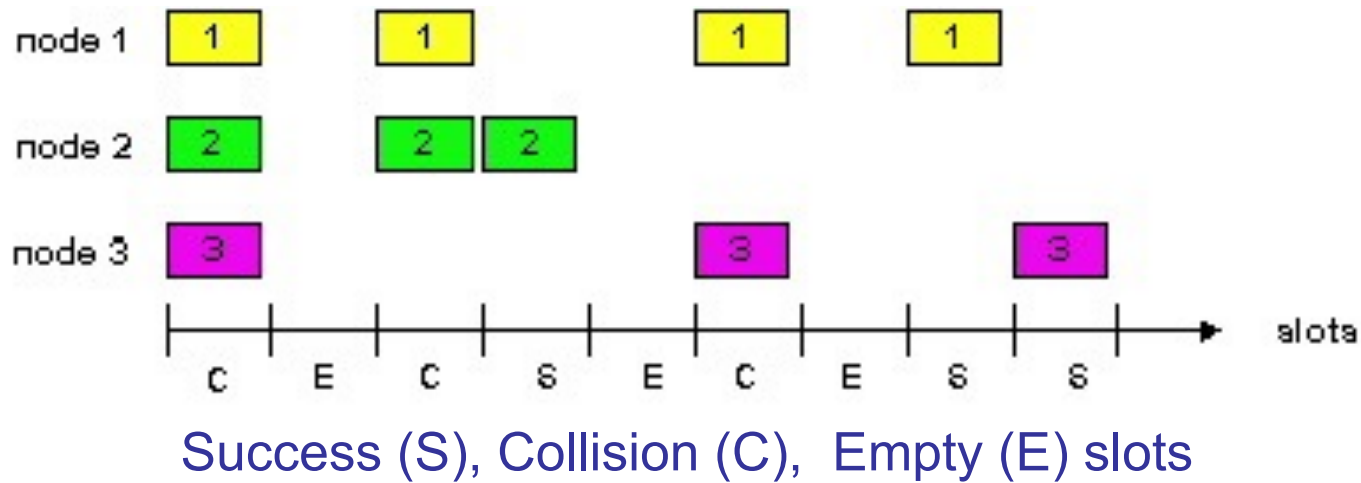
- Frame sent at t_0 collides with frames sent in $[t_0-1, t_0+1]$
 - ◆ Assuming unit-length frames
 - ◆ Ignores propagation delay





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





Channel Efficiency

Q: What is max fraction slots successful?

A: Suppose n stations have packets to send

- ◆ Each transmits in slot with probability p
- ◆ Prob[successful transmission], S , is:

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

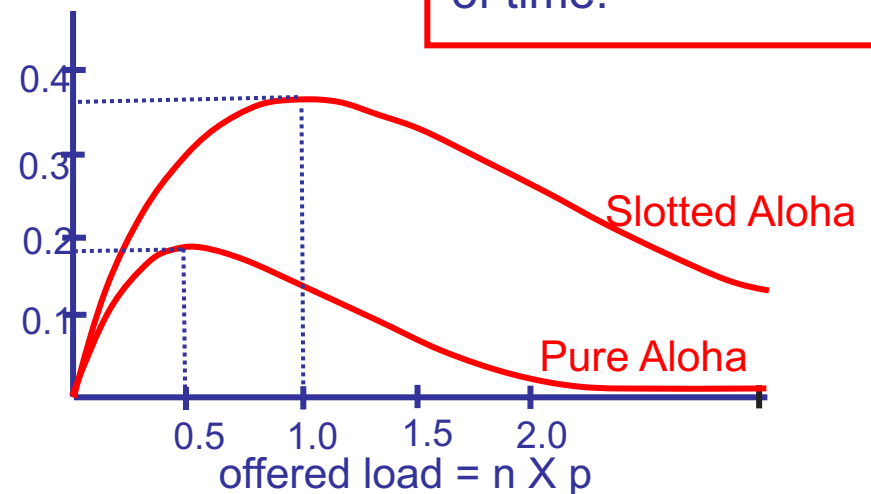
- ◆ any of n nodes:

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

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

$$= 1/e = .37$$

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





For Next Time

- Keep going on the project...