Data-Link Layers

Sangtae Ha

CSCI 4273/5273 Network Systems

http://ngn.cs.colorado.edu/~sangtaeha/courses/csci4273/fall15/

Note: The slides are adapted from the materials from Prof. Richard Han at CU Boulder and Profs. Jennifer Rexford and Mike Freedman at Princeton University.

Announcements

- PA #1 is out, due 11.59 pm Sept 10 (Thu)
- Labs
  - Create a group of three students for the lab and inform your TA your schedule
  - Lab 1: Sept. 11 (Fri), 12 (Sat), 13 (Sun), 14 (Mon)

Programming Assignment #1
Web Server

Anatomy of HTTP 1.0

Web Client

Connect: Request

Web Server

```
GET / HTTP/1.0
Host: www.yahoo.com
CRLF
```

Response: Close

```
HTTP/1.0 200 OK
Date: Tue, 16 Feb 2010 19:21:24 GMT
Content-Type: text/html;
CRLF
<html><head><title>Yahoo!</title>
```

Anatomy of HTTP 1.0

Web Client	Connect: Request	Web Server
	$\longrightarrow$	
Request Line	GET / HTTP/1.0	
Request Header	Host: www.yahoo.com	
Request Delimiter	CRLF	
	$\longleftarrow$	
	Response: Close	
Response Status	HTTP/1.0 200 OK	
Response Header	Date: Tue, 16 Feb 2010 19:21:24 GMT	
	Content-Type: text/html;	
Response Delimiter	CRLF	
Response Body	`<html><head><title>Yahoo!</title>`	

HTTP 1.1 vs 1.0

- Additional Methods (PUT, DELETE, TRACE, CONNECT + GET, HEAD, POST)
- Additional Headers
- Transfer Coding (chunk encoding)
- Persistent Connections (content-length matters)
- Request Pipelining

Hints

- Read the configuration file “ws.conf”
  - Set the server port number by reading the line (Listen 8004)
  - Set the document root directory (DocumentRoot /www)
  - … close the file “ws.conf”
- Run the server
  - socket() -> bind () -> listen() -> accept()
  - when the server read() data from the client, check the content if it is a valid HTTP request (“GET xx /HTTP 1.0 …”)
  - If the request is not valid, send the error message to the client
    - HTTP/1.0 400 Bad Request - Invalid Method, Invalid URI, Invalid HTTP-version
    - HTTP/1.0 404 Not Found: /img/logo.jpg
    - HTTP/1.0 501 Not Implemented: /video/lecture1.mpg
    - HTTP/1.0 500 Internal Server Error: cannot allocate memory

Hints (Continued)

- Run the server
  - …
  - If the request is valid, then it prepares the response
    - HTTP/1.0 200 OK
    - Date
    - Content-Type: <>
    - Content-Length: <> (if necessary)
    - CRLF
    - <File Content>
  - Close the connection (HTTP 1.0)

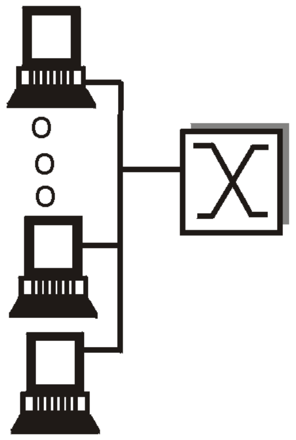
Link = Medium + Adapters

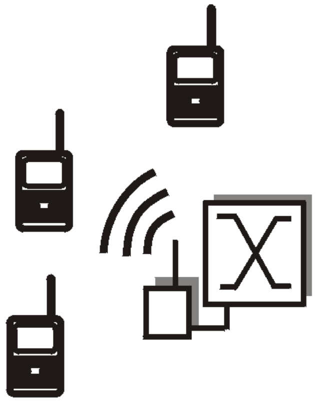
What is a Link?

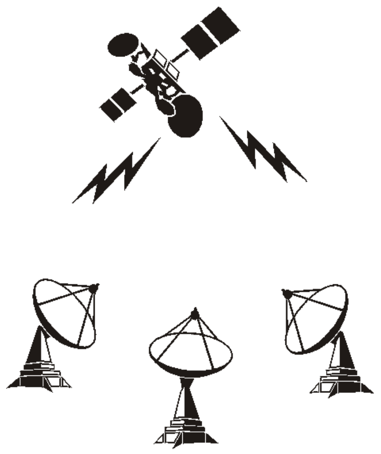
Communication Medium

Network Adapter

Broadcast Links: Shared Media

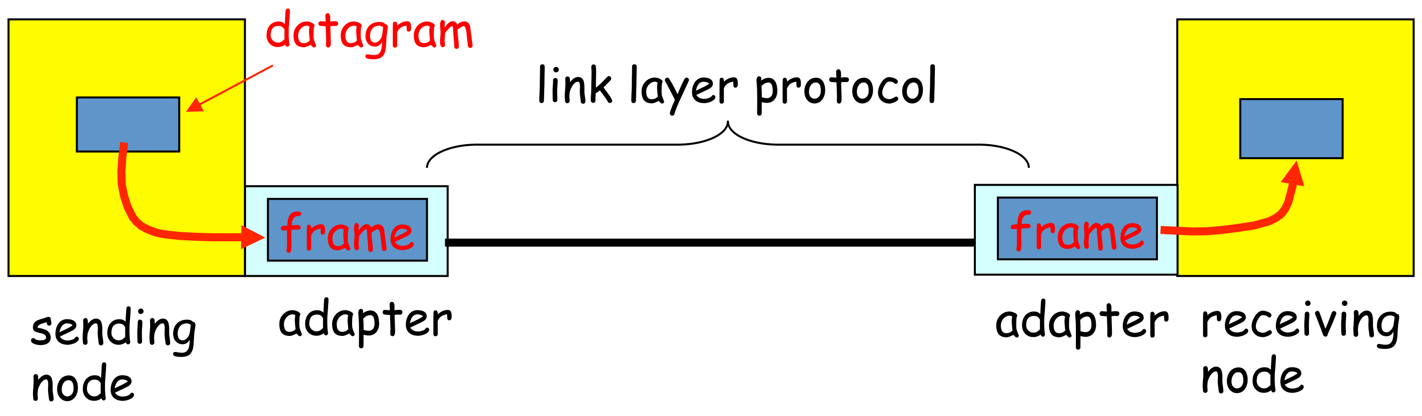

shared wire
(e.g. Ethernet)


shared wireless
(e.g. Wavelan)


satellite

cocktail party

Digital adaptors Communicating



- Link layer implemented in adaptor (network interface card)
  - Ethernet card, PCMCIA card, 802.11 card
- Sending side:
  - Encapsulates datagram in a frame
  - Adds error checking bits, flow control, etc.
- Receiving side
  - Looks for errors, flow control, etc.
  - Extracts datagram and passes to receiving node

Link-Layer Services

- Encoding
  - Representing the 0s and 1s
- Framing
  - Encapsulating packet into frame, adding header, trailer
  - Using MAC addresses, rather than IP addresses
- Error detection
  - Errors caused by signal attenuation, noise.
  - Receiver detecting presence of errors
- Error correction
  - Receiver correcting errors without retransmission
- Flow control
  - Pacing between adjacent sending and receiving nodes

Link-Layer Protocols

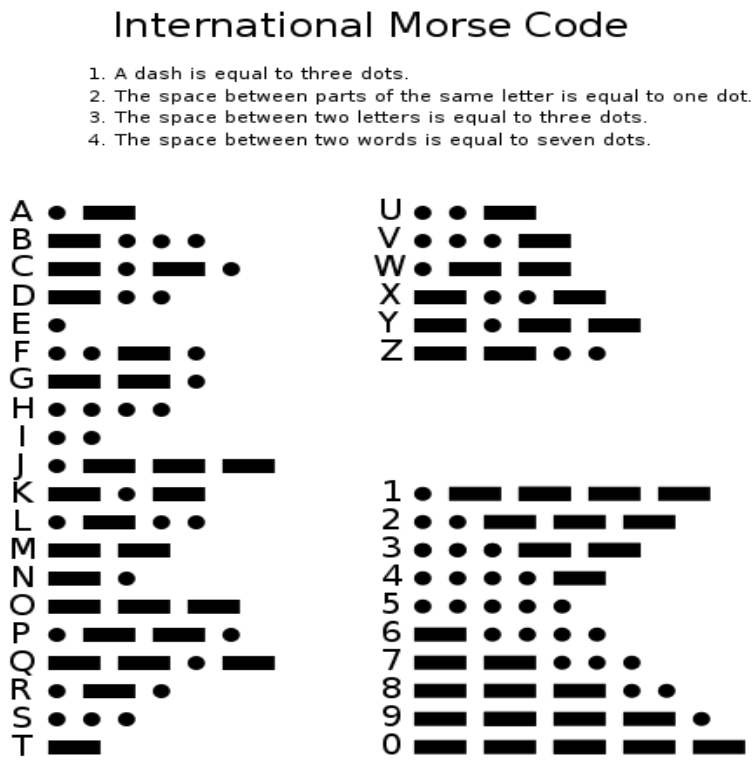
Outline

- Link-layer protocols
  - Encoding, framing, error detection
- Multiple-access links: sharing is caring!
  - Strict isolation: division over time or frequency
  - Centralized management (e.g., token passing)
  - Decentralized management (e.g., CSMA/CD)

Digital -> Analog Encoding

- Signals sent over physical links
  - Source node: bits -> signal
  - Receiving node: signal -> bits

- Encoding in telegraph
  - Morse code: “long” and “short” signals



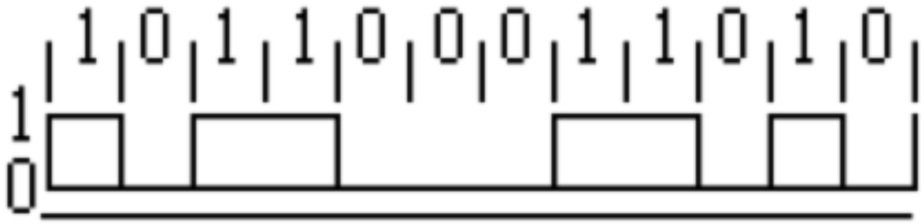
International Morse Code

1. A dash is equal to three dots.
2. The space between parts of the same letter is equal to one dot.
3. The space between two letters is equal to three dots.
4. The space between two words is equal to seven dots.

A	● ▬	U	● ● ▬
B	▬ ● ● ●	V	● ● ● ▬
C	▬ ● ▬ ●	W	● ▬ ▬
D	▬ ● ●	X	▬ ● ● ▬
E	●	Y	▬ ● ▬ ▬
F	● ● ▬ ●	Z	▬ ▬ ● ●
G	▬ ▬ ●		
H	● ● ● ●		
I	● ●		
J	● ▬ ▬ ▬		
K	▬ ● ▬	1	● ▬ ▬ ▬ ▬
L	● ▬ ● ●	2	● ● ▬ ▬ ▬
M	▬ ▬	3	● ● ● ▬ ▬
N	▬ ●	4	● ● ● ● ▬
O	▬ ▬ ▬	5	● ● ● ● ●
P	● ▬ ▬ ●	6	▬ ● ● ● ●
Q	▬ ▬ ● ▬	7	▬ ▬ ● ● ●
R	● ▬ ●	8	▬ ▬ ▬ ● ●
S	● ● ●	9	▬ ▬ ▬ ▬ ●
T	▬	0	▬ ▬ ▬ ▬ ▬

Digital -> Analog Encoding

- Signals sent over physical links
  - Source node: bits -> signal
  - Receiving node: signal -> bits
- Simplify some electrical engineering details
  - Assume two discrete signals, high and low
  - E.g., could correspond to two different voltages
- Simple approach: Non-return to zero
  - High for a 1, low for a 0

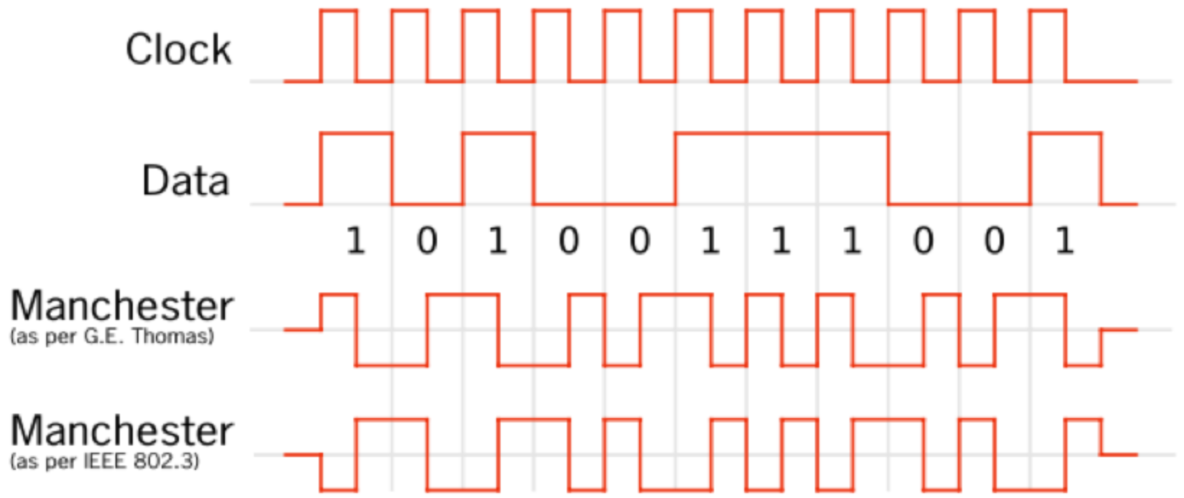


Problem With NRZ

- Long strings of 0s or 1s introduce problems
  - No transitions from low-to-high, or high-to-low
- Receiver keeps average of signal it has received
  - Uses the average to distinguish between high and low
  - Long flat strings make receiver sensitive to small change
- Transitions also necessary for clock recovery
  - Receiver uses transitions to derive its own clock
  - Long flat strings do not produce any transitions
  - Can lead to clock drift at the receiver
- Alternatives (see Section 2.2)
  - Non-return to zero inverted, and Manchester encoding

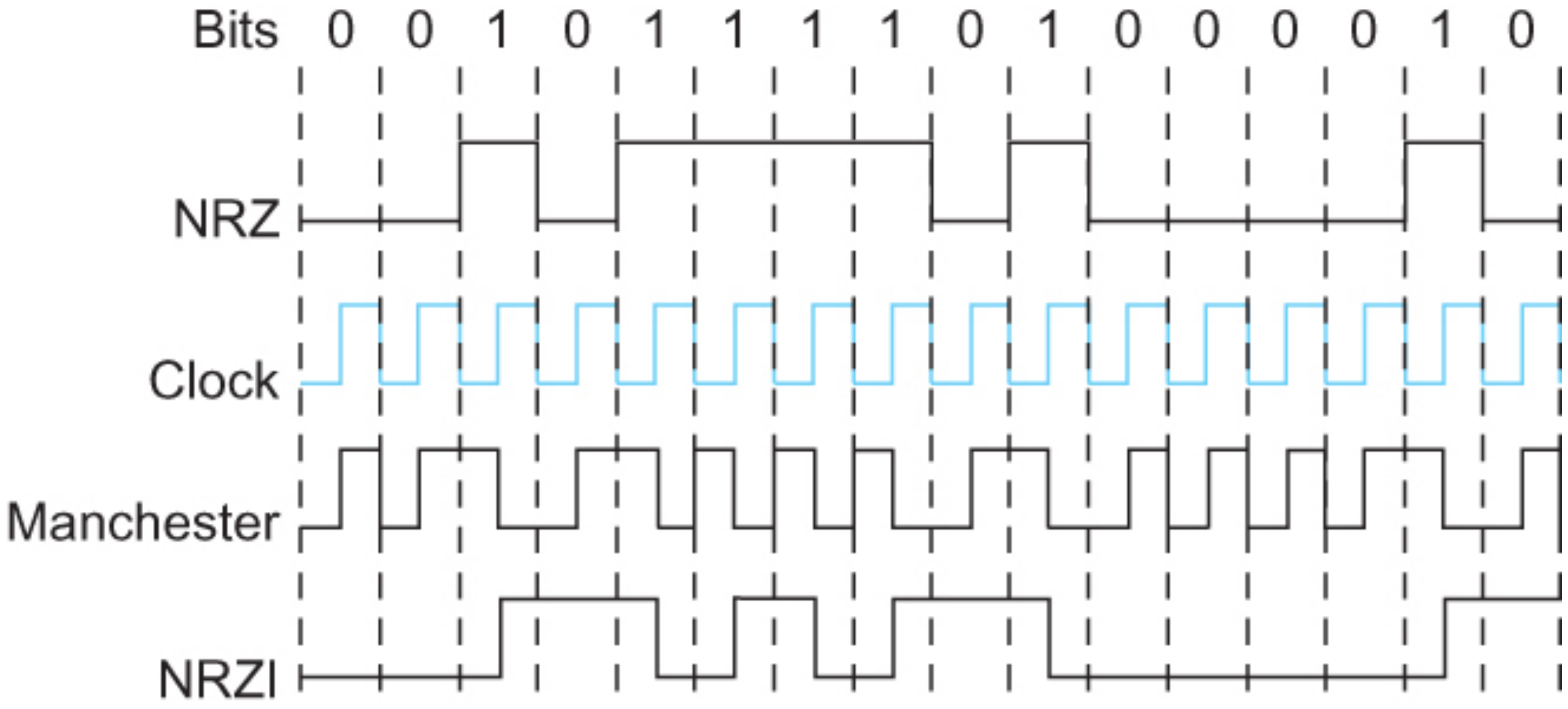
Protocols with clock-recovery

- Manchester encoding (basic Ethernet)
  - clock XOR NRZ: 802.3: H→L (0), L→H (1) : self-clocking



- Efficiency? (read 4B/5B encoding in Sec 2.2)
  - Manchester: 2 clock transitions per bit: 50% efficient
  - 1 GbE: 8b/10b: 80% efficient
  - 10 GbE: 64b/66b: 96.9% efficient

Different Encoding Strategies



Framing

- Break sequence of bits into a frame
  - Typically implemented by the network adaptor
- Sentinel-based
  - Delineate frame with special pattern (e.g., 01111110)

`01111110`	`Frame contents`	`01111110`

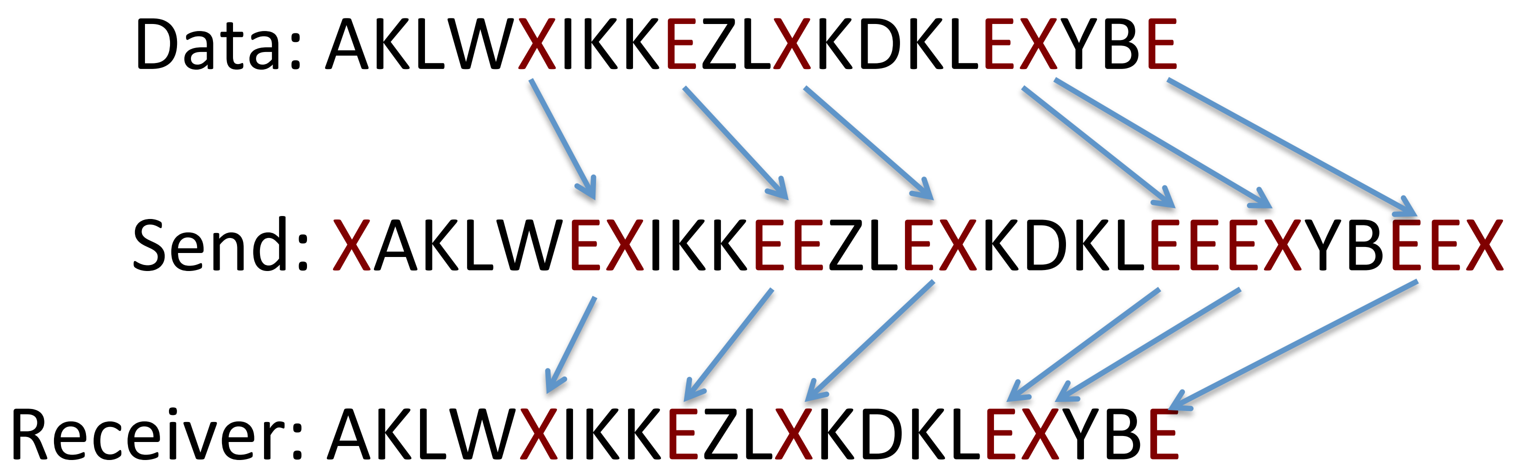
  - Problem: what if special patterns occurs within frame?
  - Solution: escaping the special characters (stuffing)
    - E.g., sender always inserts a 0 after five 1s
    - … and receiver always removes a 0 appearing after five 1s
    - Byte stuffing (BiSync, PPP) and bit stuffing (HDLC)
  - Similar to escaping special characters in C programs

Framing (Continued)

- Counter-based
  - Include the payload length in the header
  - … instead of putting a sentinel at the end
  - Problem: what if the count field gets corrupted?
    - Causes receiver to think the frame ends at a different place
  - Solution: catch later when doing error detection
    - And wait for the next sentinel for the start of a new frame
- Clock-based
  - Make each frame a fixed size
  - No ambiguity about start and end of frame
  - But, may be wasteful

Character/Byte Stuffing Example

- Sentinel X, at both start and end of packet
- Stuff X in data, replace X with escape character "E" (DLE in textbook) and X, i.e., X -> (E,X)



Data: AKLWXIKKEZLXKDKLEXYBE

Send: XAKLWEXIKKEEZLEXKDKLEEEXYBEEX

Receiver: AKLWXIKKEZLXKDKLEXYBE

Bit Stuffing Example

- Similar to byte stuffing, except bit stuffing is not confined to byte boundaries
- HDLC denotes beginning and end of a packet/frame with "01111110" flag
- Since "01111110" may occur anywhere (across byte boundaries) in data, then "stuff" it:
  - At sender, after 5 consecutives one, insert a "0"
  - At receiver, "0111110" -> stuffing, so destuff, "01111110" -> end of frame, "01111111" -> error

```
    Data:  0110111111100111110111111111100000

    Send:  01101111101100111110011111011111000000

Receiver:  0110111111100111110111111111100000
```

Error Detection

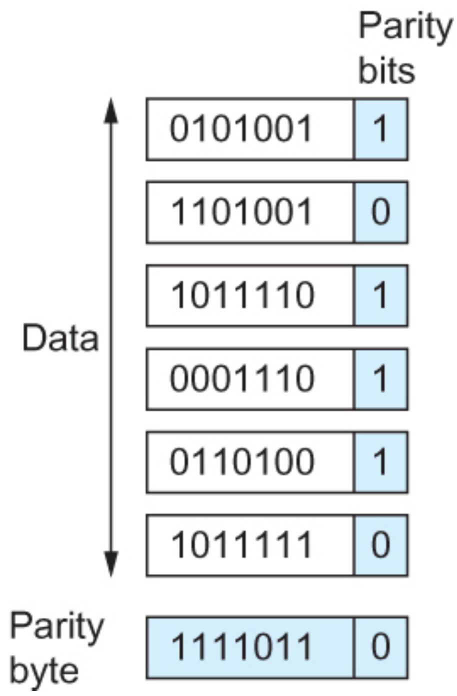
- Errors are unavoidable
  - Electrical interference, thermal noise, etc.
- Error detection
  - Transmit extra (redundant) information
  - Use redundant information to detect errors
  - Extreme case: send two copies of the data
  - Trade-off: accuracy vs. overhead

Probability of Packet Error

- Send N bits. Probability of bit error = P_b
- Assume independent bit errors. What is the probability of packet error?
  - Prob[packet error] = Prob[at least 1 bit is corrupt] = 1- Prob[every bit is clean] = $1-(1-P_b)^N$
  - If $P_b = 10^{-6}$, and N=10000 bits, then Prob[packet error] = $9.95 * 10^{-3}$ ~= 1%
- Optical links have much lower probabilities of bit error: 10^{-12}
- Wireless links have much higher probabilities of bit error: 10^{-3}
  - Bit errors correlated, not independent

Error Detection Techniques

- Parity check
  - Add an extra bit to a 7-bit code
  - Odd parity: ensure an odd number of 1s
    - E.g., 0101011 becomes 01010111
  - Even parity: ensure an even number of 1s
    - E.g., 0101011 becomes 01010110
  - Two dimensional parity



- Checksum
  - Treat data as a sequence of 16-bit words
  - Compute a sum of all 16-bit words, with no carries
  - Transmit the sum along with the packet
- Cyclic Redundancy Check (CRC)
  - See Section 2.4.3

Error Correction

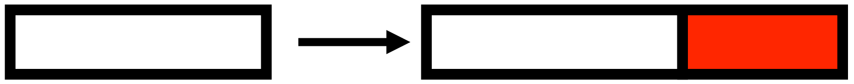
- Correct an error, rather than just detecting an error
- Simple technique: Send 2 copies of data with the data (repetition coding), and then do majority logic decoding at the receiver
  - Original data: 0100
  - Send: 0100 0100 0100
- Problems
  - In efficient
  - It cannot correct 2 errors in the same bit

Receiver: 0100
0110
0100
Decode: 0100

Corrupt Bit

Error Correction (Continued)

- Forward Error Correction (FEC)
  - Many types, e.g., Reed-Solomon coding
  - Add K bits of redundancy to N bits, to form a (N+K)-bit long packet, or vector



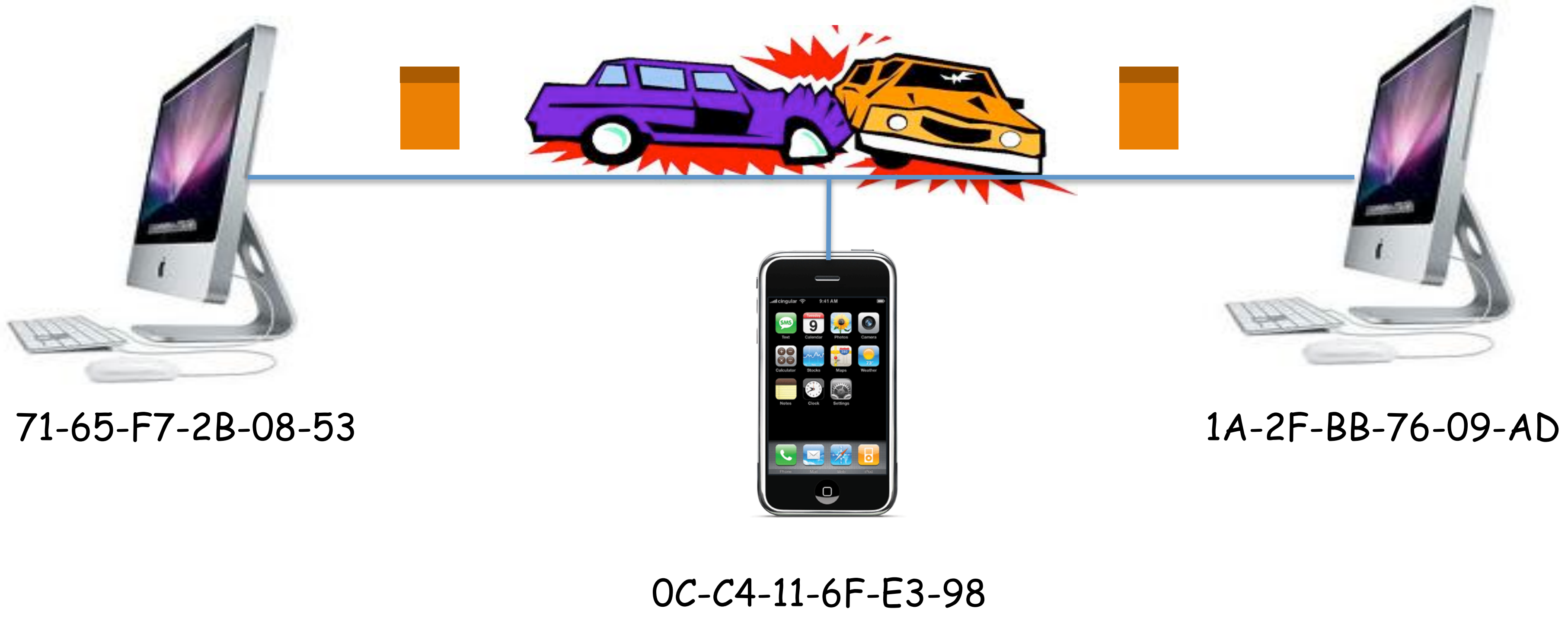
  - N dimensions -> N+K dimensions
  - 2N patterns or vectors mapped into 2N+K possibilities
  - Spread out these vectors as far away from neighbors as possible in (N+K)-dimensional space
  - N=2, K=3, N+K = 5

11	⟶	11111
10	⟶	01010
01	⟶	10101
00	⟶	00000

  - Receive 01111 – closet to 11111, so decode “11” and correct one bit error

Sharing the Medium

Collisions



- Single shared broadcast channel
  - Avoid having multiple nodes speaking at once
  - Otherwise, collisions lead to garbled data

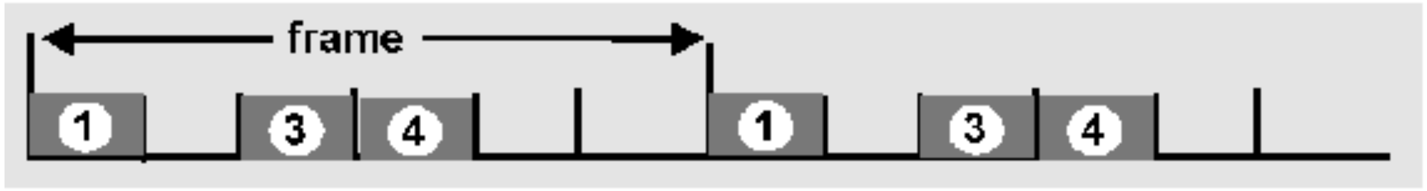
Multiple Access Protocol

- Single shared broadcast channel
  - Avoid having multiple nodes speaking at once
  - Otherwise, collisions lead to garbled data
- Multiple access protocol
  - Distributed algorithm for sharing the channel
  - Algorithm determines which node can transmit
- Classes of techniques
  - Channel partitioning: divide channel into pieces
  - Taking turns: passing a token for the right to transmit
  - Random access: allow collisions, and then recover

Channel Partitioning: TDMA

TDMA: time division multiple access

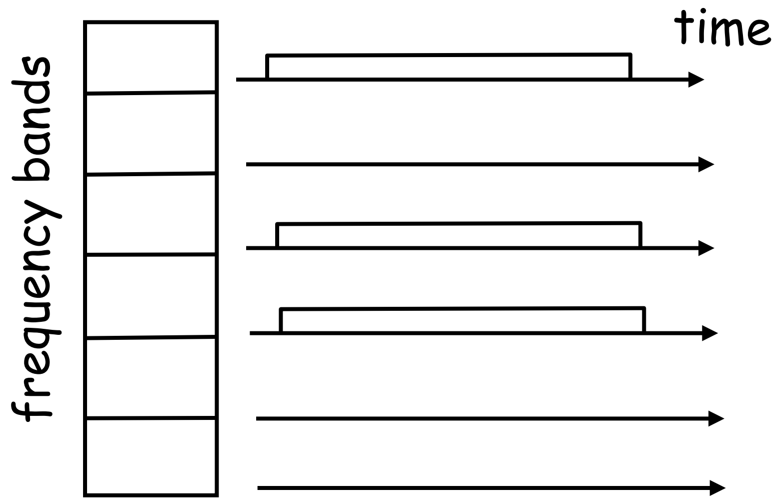
- Access to channel in "rounds"
  - Each station gets fixed length slot in each round
- Time-slot length is packet transmission time
  - Unused slots go idle
- Example: 6-station LAN with slots 1, 3, and 4



Channel Partitioning: FDMA

FDMA: frequency division multiple access

- Channel spectrum divided into frequency bands
  - Each station has fixed frequency band (Wifi channels 1-11)
- Unused transmission time in bands go idle
- Example: 6-station LAN with bands 1, 3, and 4



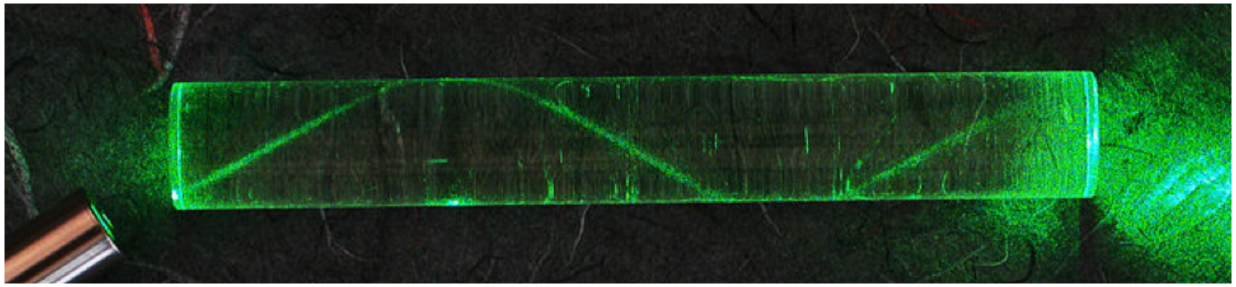
Channel Partitioning: FDMA

FDMA: frequency division multiple access

- Channel spectrum divided into frequency bands
  - Each station has fixed frequency band (Wifi channels 1-11)
- Unused transmission time in bands go idle
- Example: 6-station LAN with bands 1, 3, and 4

WDM: Wavelength division multiplexing

- Multiple wavelengths λ on same optical fiber



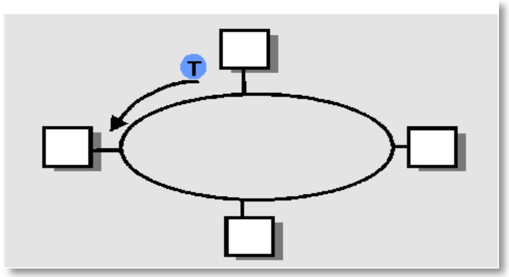
"Taking Turns" MAC protocols

Polling

- Primary node "invites" secondary nodes to transmit in turn
- Concerns:
  - Polling overhead
  - Latency
  - Single point of failure (primary)

Token passing

- Control token passed from one node to next sequentially
- Token message
- Concerns:
  - Token overhead
  - Latency
  - Single point of failure (token)



Random Access Protocols

- When node has packet to send
  - Transmit at full channel data rate R.
  - No *a priori* coordination among nodes
- Two or more transmitting nodes ➔ "collision"
- Random access MAC protocol specifies:
  - How to detect collisions
  - How to recover from collisions

Key Ideas of Random Access

- Carrier Sense (CS)
  - *Listen before speaking, and don't interrupt*
  - Checking if someone else is already sending data
  - … and waiting till the other node is done
- Collision Detection (CD)
  - *If someone else starts talking at the same time, stop*
  - Realizing when two nodes are transmitting at once
  - …by detecting that the data on the wire is garbled
- Randomness
  - *Don't start talking again right away*
  - Waiting for a random time before trying again

ALOHA Protocol

- "pure" ALOHA: hosts transmit whenever they have information to send – form of random access
  - Collision will occur when two hosts try to transmit packets at the same time
  - Hosts wait a timeout=1RTT for an ACK
  - If no ACK by timeout, then wait a randomly selected delay to avoid repeated collision, then retransmit
- Collision of packets can occur when packet overlaps another packet
  - Wasted time due to a collision = up to 2 packet intervals
  - Maximum throughput is around 18% (Poisson packet arrival)
- How to improve?
  - How about using a fixed time slot? Slotted ALOHA

Slotted ALOHA

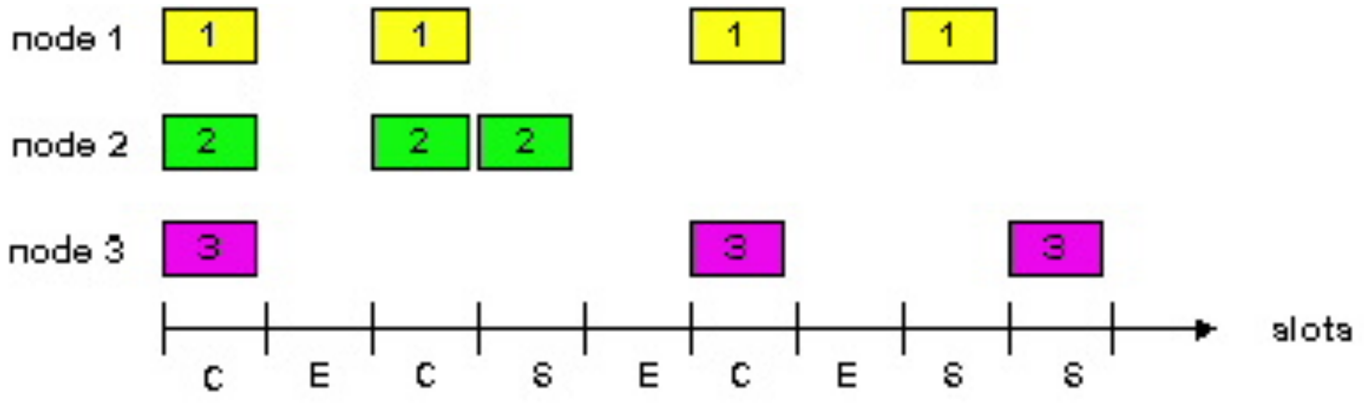
Assumptions

- All frames same size
- Time divided into equal slots (time to transmit a frame)
- Nodes start to transmit frames only at start of slots
- Nodes are synchronized
- If two or more nodes transmit, all nodes detect collision

Operation

- When node obtains fresh frame, transmits in next slot
- No collision: node can send new frame in next slot
- Collision: node retransmits frame in each subsequent slot with probability *p* until success

Slotted ALOHA



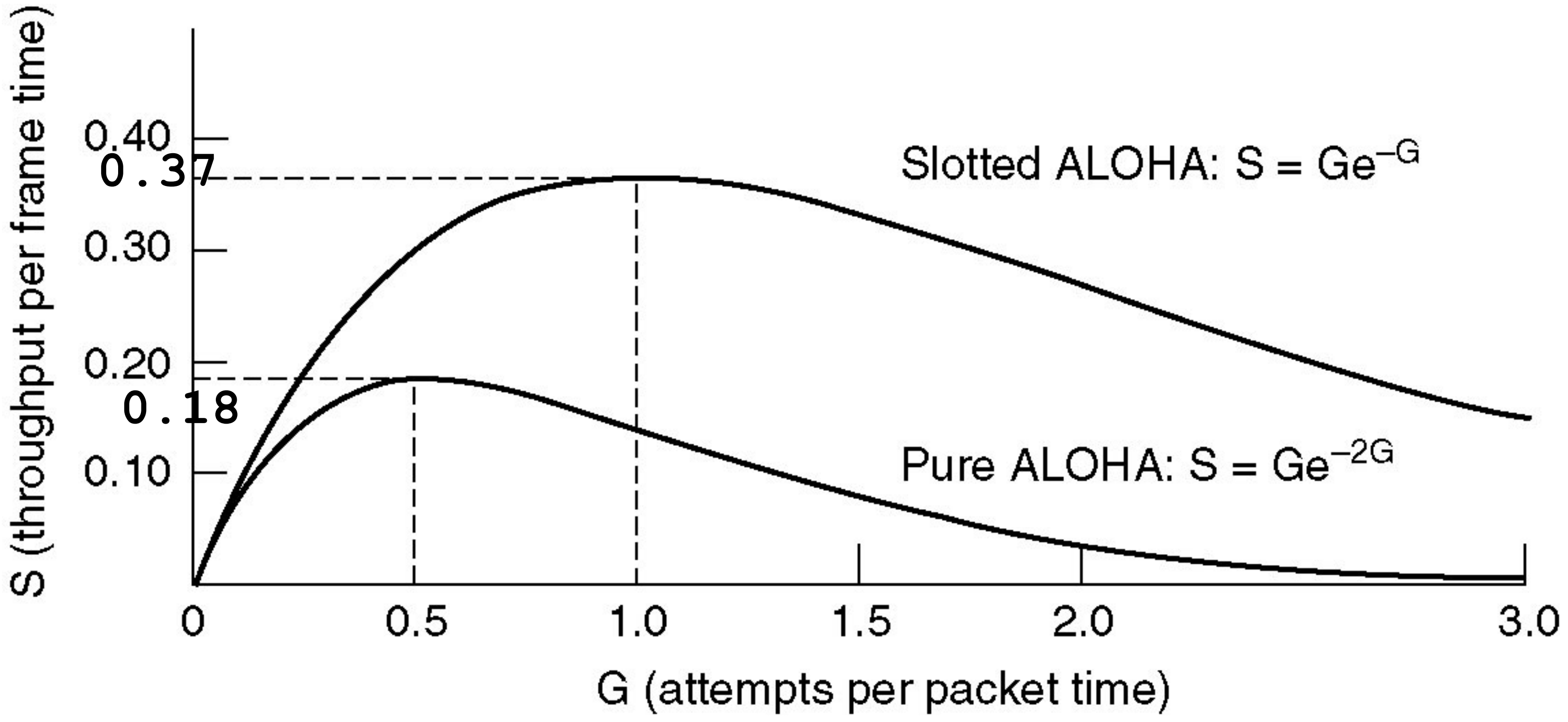
Pros

- Single active node can continuously transmit at full rate of channel
- Highly decentralized: only slots in nodes need to be in sync
- Simple

Cons

- Collisions, wasting slots
- Idle slots
- Nodes may be able to detect collision in less than time to transmit packet
- Clock synchronization

ALOHA vs. Slotted ALOHA



G is a measure of the aggregate offered load

CSMA (Carrier Sense Multiple Access)

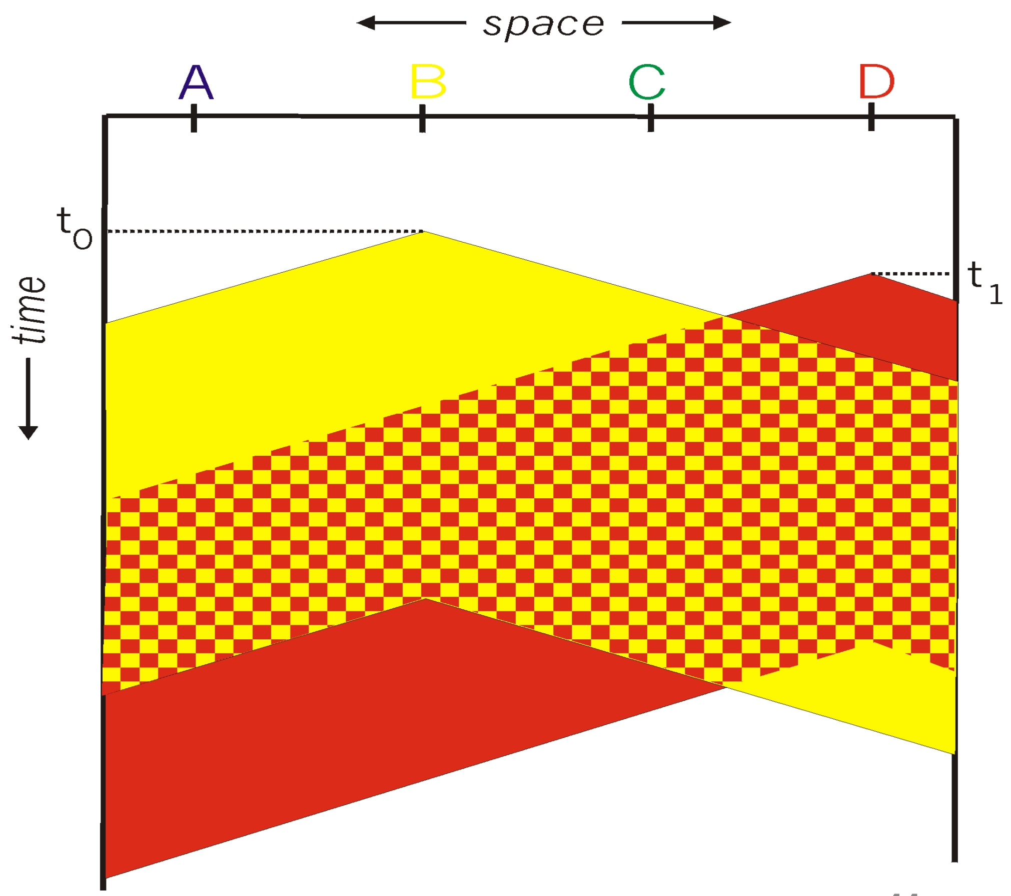
- Collisions hurt the efficiency of ALOHA protocol
  - At best, channel is useful 37% of the time
- ALOHA: transmit before listen
- CSMA: listen before transmit
  - If channel sensed idle: transmit entire frame
  - If channel sensed busy, defer transmission
- Human analogy: don't interrupt others!

CSMA (Carrier Sense Multiple Access)

CSMA: Listen before transmit

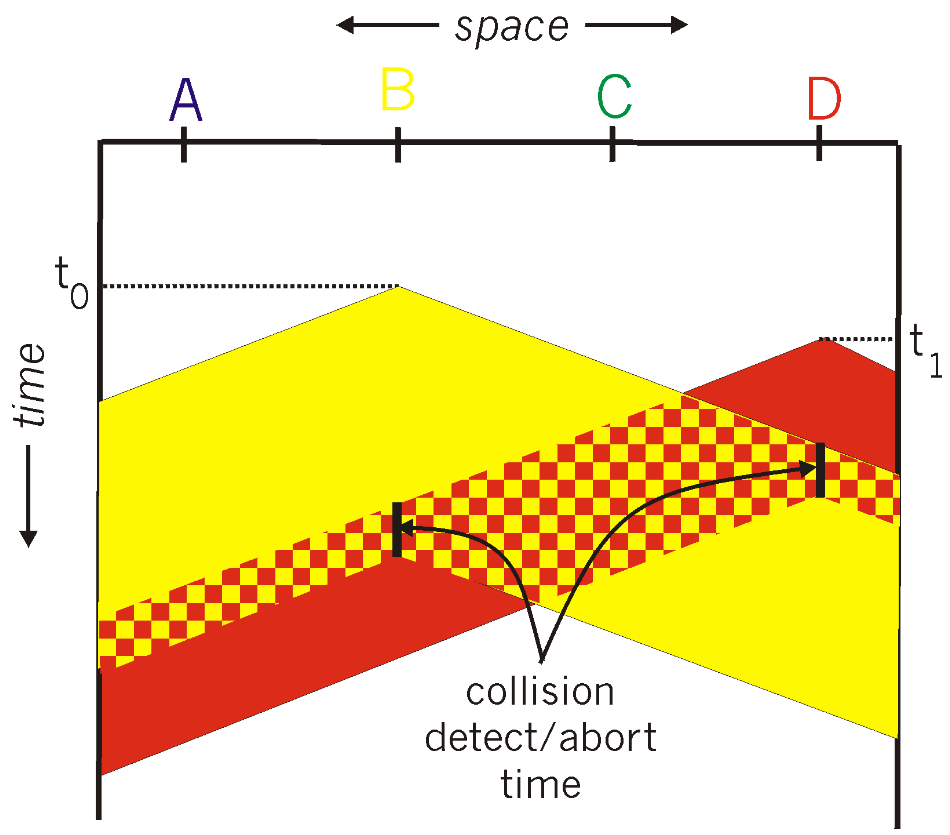
Collisions *can* still occur: propagation delay means two nodes may not hear each other's transmission

Collision: entire packet transmission time wasted



CSMA/CD Collision Detection

- Detect collision
  - Abort transmission
  - Jam the link
- Wait random time
  - Transmit again
- Hard in wireless
  - Must receive data while transmitting



Three Ways to Share the Media

- Channel partitioning MAC protocols:
  - Share channel efficiently and fairly at high load
  - Inefficient at low load: delay in channel access, 1/N bandwidth allocated even if only 1 active node!
- "Taking turns" protocols
  - Eliminates empty slots without causing collisions
  - Vulnerable to failures (e.g., failed node or lost token)
- Random access MAC protocols
  - Efficient at low load: single node can fully utilize channel
  - High load: collision overhead

Comparing the Three Approaches

- Channel partitioning is

  (a) Efficient/fair at high load, inefficient at low load

  (b) Inefficient at high load, efficient/fair at low load

- "Taking turns"

  (a) Inefficient at high load

  (b) Efficient at all loads

  (c) Robust to failures

- Random access

  (a) Inefficient at low load

  (b) Efficient at all load

  (c) Robust to failures

Comparing the Three Approaches

- Channel partitioning is
  (a) Efficient/fair at high load, inefficient at low load
  (b) Inefficient at high load, efficient/fair at low load
- "Taking turns"
  (a) Inefficient at high load
  (b) Efficient at all loads
  (c) Robust to failures
- Random access
  (a) Inefficient at low load
  (b) Efficient at all load
  (c) Robust to failures