Link Layer - Principles

Link Layer Part 1

Link Layer

- Goal
- Terminology
- Services

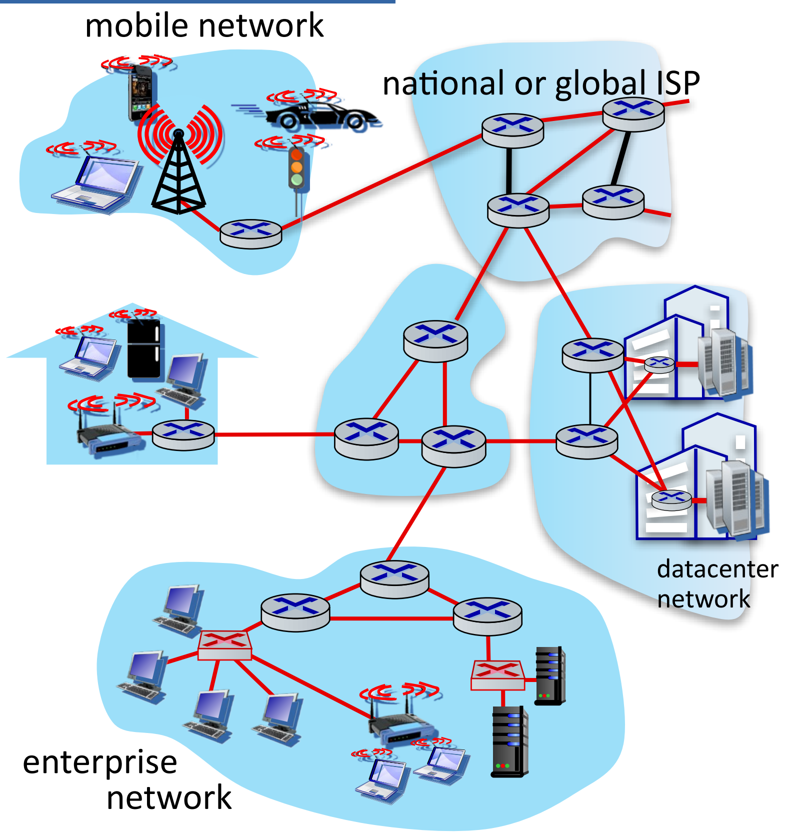
Goal

link layer has responsibility of transferring datagram from one node to physically adjacent node over a link

Terminology

terminology:

- hosts and routers: nodes
- communication channels that connect adjacent nodes along communication path: links
  - wired
  - wireless
  - LANs
- layer-2 packet: *frame*, encapsulates datagram

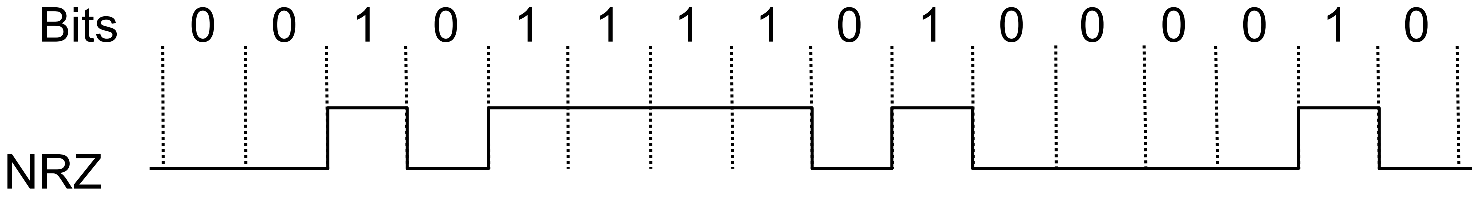


Link Layer Services

- Convert bits to signals and recover bits from received signals
  - Encoding
- Decide on a minimum unit for sending bits
  - Cannot send bit by bit (too much overhead)
  - Frame creation
- Error detection and/or correction of frames
- Multiple Access Protocols (MAP)
- Flow control
  - ARQ, Sliding WINDOW
- Addressing
  - MAC address

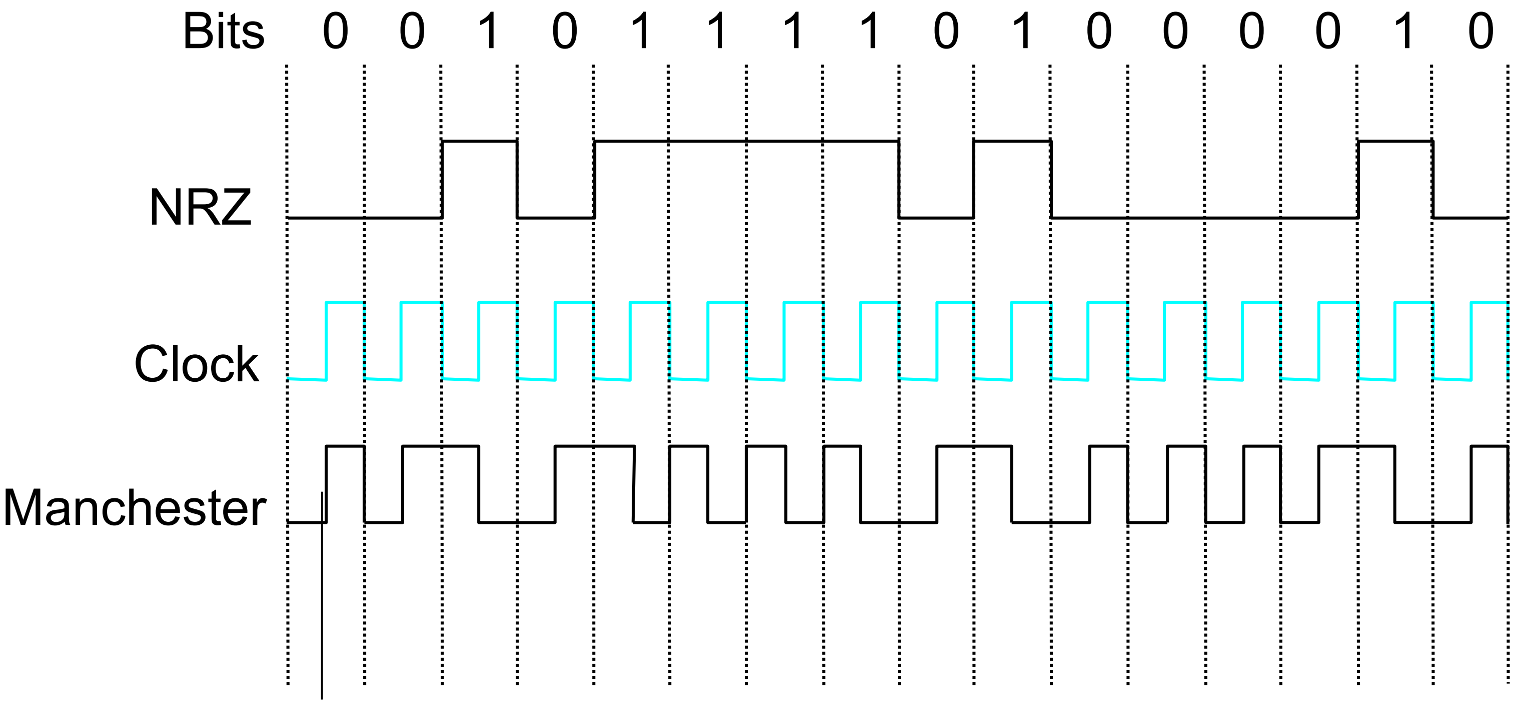
Encoding

- Signals propagate over a physical medium
  - modulate electromagnetic waves
  - e.g., vary voltage
- Encode binary data onto signals
  - e.g., 0 as low signal and 1 as high signal
  - known as Non-Return to zero (NRZ)
  - Problem: consecutive 1s and 0s , noise levels



5V 3 1s or 4 1s

Encoding



- Manchester encoding: +ve transition → 0; -ve transition → 1
- XOR(bit,clock)

Framing

- The data unit at the data link layer is called a "frame"
- A frame is a group of bits, typically in sequence
- Issues:
  - Frame creation
    - How many bits (size of frame)
    - Overhead
  - Frame delineation
  - Have meta tags
    - start and stop characters or bit sequence
  - What if the meta tags appear in the message?

Framing

- Character stuffing ($# =BOF, $^=EOF)
  - $# this prof is good $^ $^ $^
  - $# this prof s$^cks $^ .. Meta tag in message
    - $# this prof s$$^cks $^ at sender
    - $# this prof s$^cks $^ at receiver, remove stuffing
- Bit stuffing: have a unique bit sequence (0x7E)
  - 01111110 this prof is good 01111110
  - 01111110 this prof is 01111110 good 01111110
  - 01111110 this prof is 011111010 good 0111110 – sender
  - Sender inserts a zero after seeing 5 consecutives 1s
  - Receiver checks for 5 1s, if next bit is 0 – stuff
  - Else If next 2 bits are 10 end of frame else error
  - 01111110 → 011111010 Sender
  - 011111010 → 01111110 ….. 01111110 -→ EOF

Multiple Access Protocol

Multiple Access Protocol

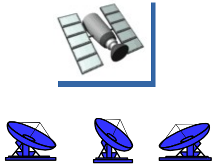
two types of “links”:

- point-to-point
  - point-to-point link between Ethernet switch, host
  - PPP for dial-up access
- broadcast (shared wire or medium)
  - old-fashioned Ethernet
  - upstream HFC in cable-based access network
  - 802.11 wireless LAN, 4G/4G. satellite

shared wire (e.g., cabled Ethernet)

shared radio: 4G/5G

shared radio: WiFi



shared radio: satellite

ail party (shared air, acoustical)

Multiple Access Protocol

- single shared broadcast channel
- two or more simultaneous transmissions by nodes: interference
  - *collision* if node receives two or more signals at the same time

multiple access protocol

distributed algorithm that determines how nodes share channel, i.e., determine when node can transmit

communication about channel sharing must use channel itself!

no out-of-band channel for coordination

Multiple Access Protocol

given: multiple access channel (MAC) of rate *R* bps

desiderata:

1. when one node wants to transmit, it can send at rate *R*.
2. when M nodes want to transmit, each can send at average rate *R/M*
3. fully decentralized:
   - no special node to coordinate transmissions
   - no synchronization of clocks, slots
4. simple

Multiple Access Protocol

three broad classes:

- channel partitioning
  - divide channel into smaller “pieces” (time slots, frequency, code)
  - allocate piece to node for exclusive use
- *random access*
  - channel not divided, allow collisions
  - “recover” from collisions
- “taking turns”
  - nodes take turns, but nodes with more to send can take longer turns

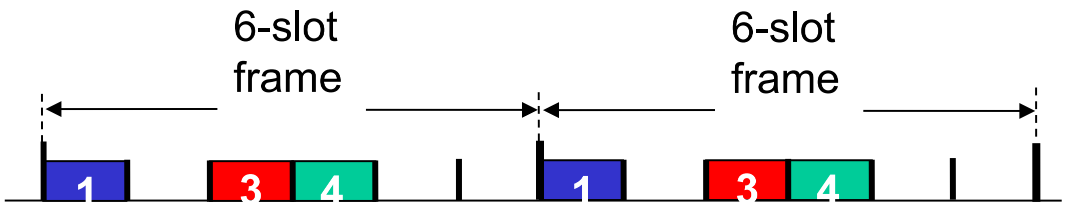
Channel Partitioning

- TDMA (Time Division Multiple Access)
- FDMA (Frequency Division Multiple Access)
- CDMA (Code Division Multiple Access)

TDMA

TDMA: time division multiple access

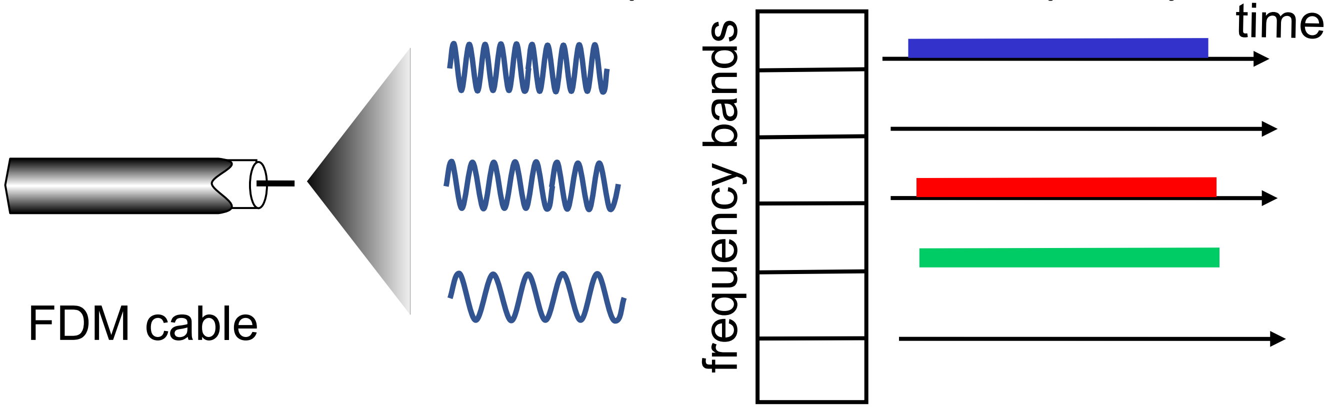
- access to channel in “rounds”
- each station gets fixed length slot (length = packet transmission time) in each round
- unused slots go idle
- example: 6-station LAN, 1,3,4 have packets to send, slots 2,5,6 idle



FDMA

FDMA: frequency division multiple access

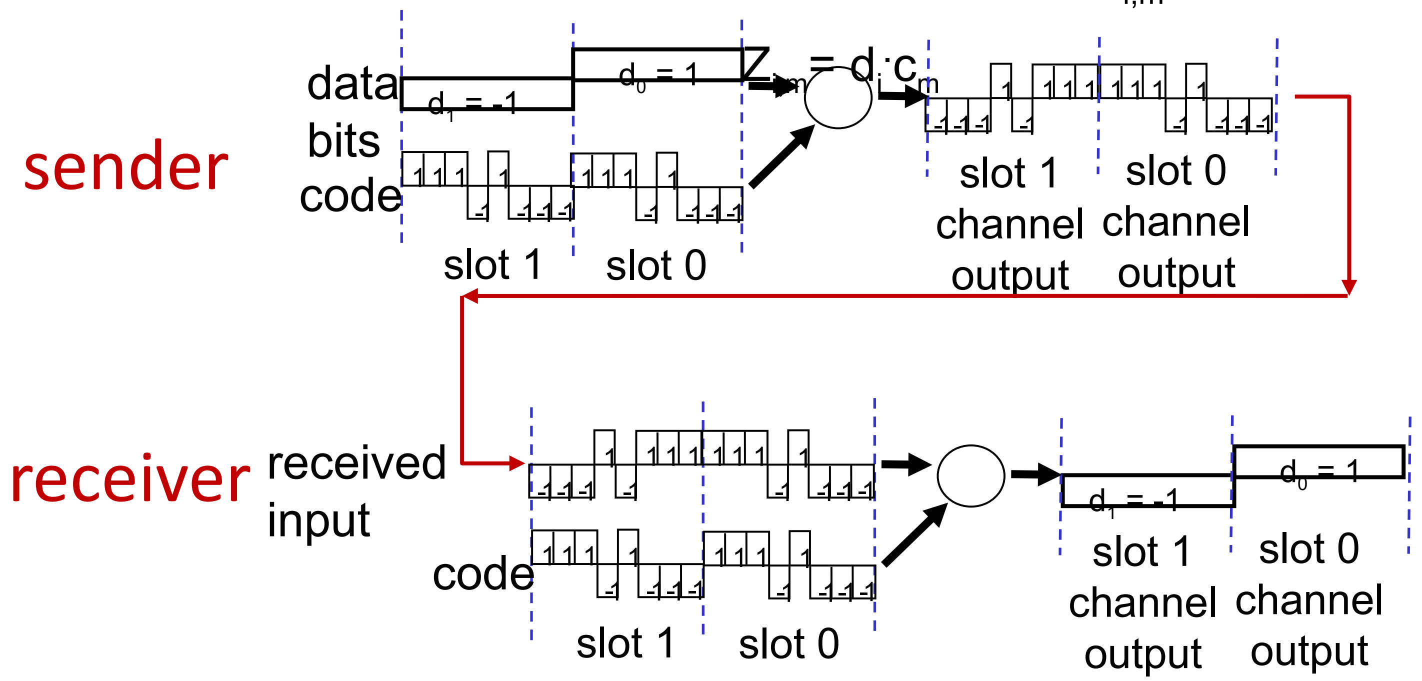
- channel spectrum divided into frequency bands
- each station assigned fixed frequency band
- unused transmission time in frequency bands go idle
- example: 6-station LAN, 1,3,4 have packet to send, frequency bands 2,5,6 idle



Code Division Multiple Access (CDMA)

- unique "code" assigned to each user; i.e., code set partitioning
  - all users share same frequency, but each user has own "chipping" sequence (i.e., code) to encode data
  - allows multiple users to "coexist" and transmit simultaneously with minimal interference (if codes are "orthogonal")
- encoding: inner product: (original data) X (chipping sequence)
- decoding: summed inner-product: (encoded data) X (chipping sequence)

CDMA



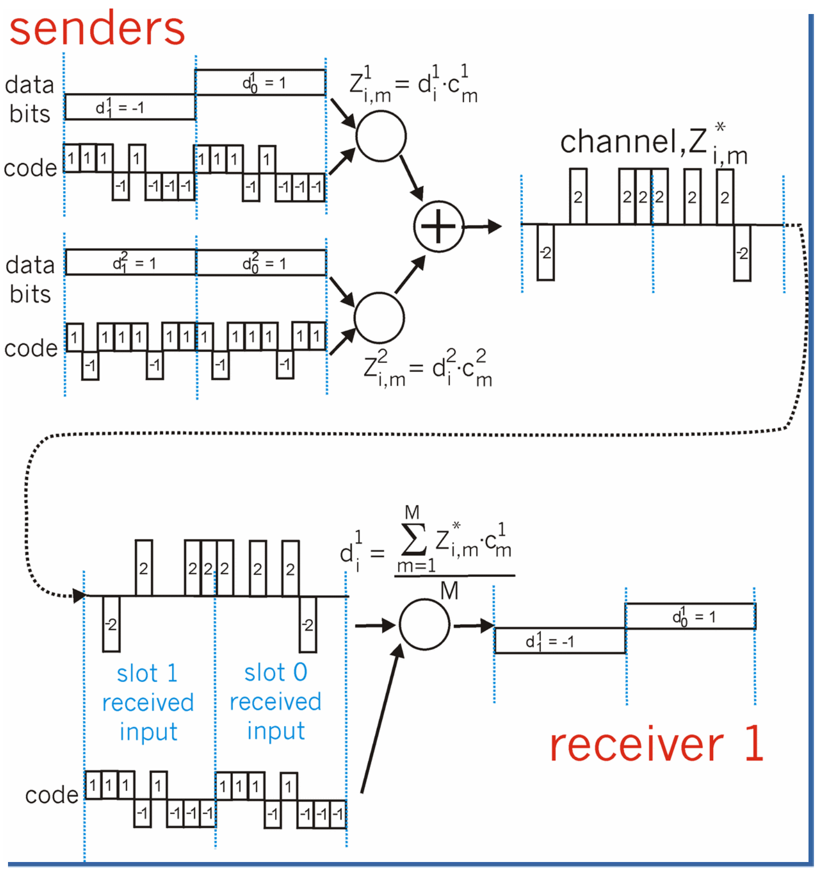
$$D_i = \frac{\sum_{m=1}^{M} Z_{i,m} \cdot c_m}{M}$$

... but this isn't really useful yet!

CDMA

Sender 1

Sender 2



channel sums together transmissions by sender 1 and 2

… now *that's* useful!

using same code as sender 1, receiver recovers sender 1's original data from summed channel data!

Random Access

ALOHA

Slotted ALOHA

CSMA

CSMA/CD

ALOHA Additive Links On-line Hawaii Area

- Originally developed for ground-based packet radio communications in 1970
- Goal: let users transmit whenever they have something to send

ALOHA

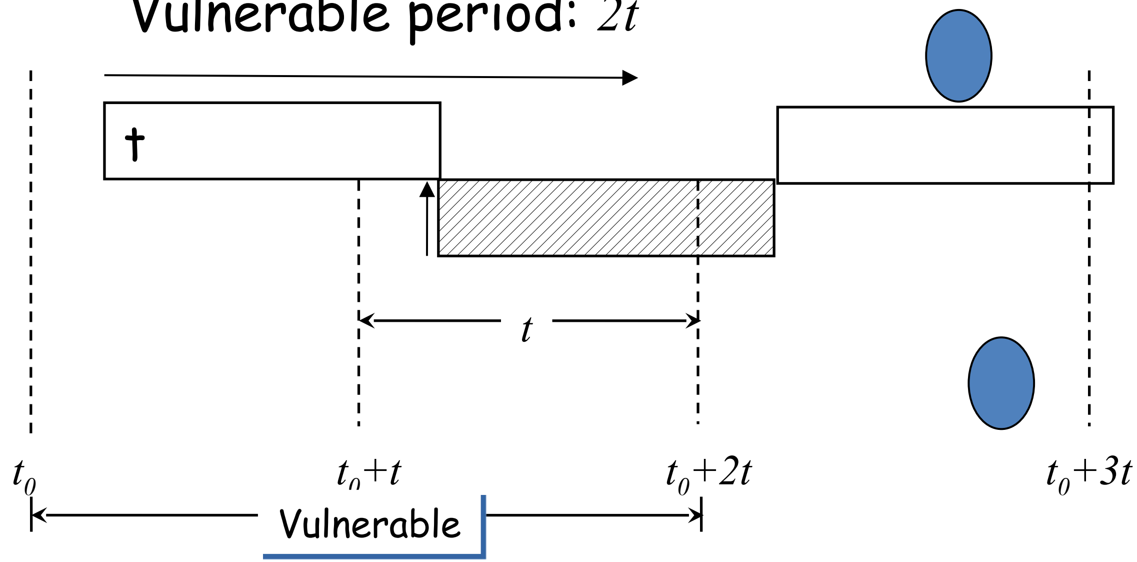
1. Transmit whenever you have data to send
2. Listen to the broadcast
   - Because broadcast is fed back, the sending host can always find out if its packet was destroyed just by listening to the downward broadcast one round-trip time after sending the packet
3. If the packet was destroyed, wait a random amount of time and send it again
   - The waiting time must be random to prevent the same packets from colliding over and over again

ALOHA

- Note that if the first bit of a new packet overlaps with the last bit of a packet almost finished, both packets are totally destroyed.

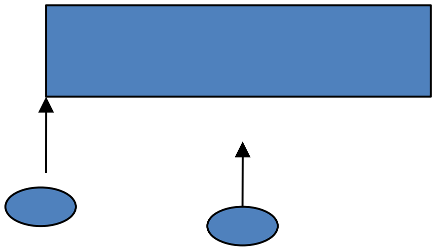
t : one packet transmission time

Vulnerable period: $2t$



ALOHA

- Due to collisions and idle periods, pure ALOHA is limited to approximately 18% throughput in the best case
- Can we improve this?



Slotted ALOHA

assumptions:

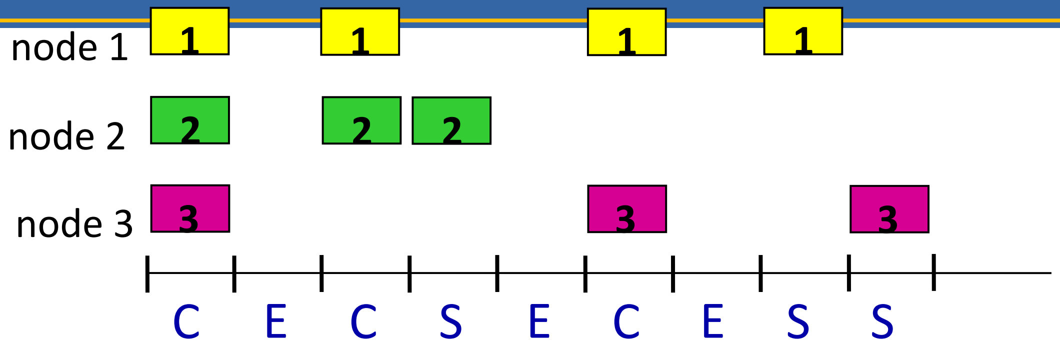
- all frames same size
- time divided into equal size slots (time to transmit 1 frame)
- nodes start to transmit only slot beginning
- nodes are synchronized
- if 2 or more nodes transmit in slot, all nodes detect collision

operation:

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

randomization – *why*?

Slotted ALOHA



C: collision
S: success
E: empty

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

Slotted ALOHA

efficiency: long-run fraction of successful slots (many nodes, all with many frames to send)

- *suppose:* N nodes with many frames to send, each transmits in slot with probability p
  - prob that given node has success in a slot $= p(1-p)^{N-1}$
  - prob that *any* node has a success $= Np(1-p)^{N-1}$
  - max efficiency: find p^* that maximizes $Np(1-p)^{N-1}$
  - for many nodes, take limit of $Np^*(1-p^*)^{N-1}$ as N goes to infinity, gives:

  max efficiency = 1/e = .37

- *at best:* channel used for useful transmissions 37% of time!

ALOHA

P(success by given node) = P(node transmits)

P(no other node transmits in $[t_0-1,t_0]$

P(no other node transmits in $[t_0-1,t_0]$

$$= p \cdot (1-p)^{N-1} \cdot (1-p)^{N-1}$$

$$= p \cdot (1-p)^{2(N-1)}$$

… choosing optimum p and then letting *n* go to infinity

$= 1/(2e) = .18$

even worse than slotted Aloha!

CSMA (carrier sense multiple access)

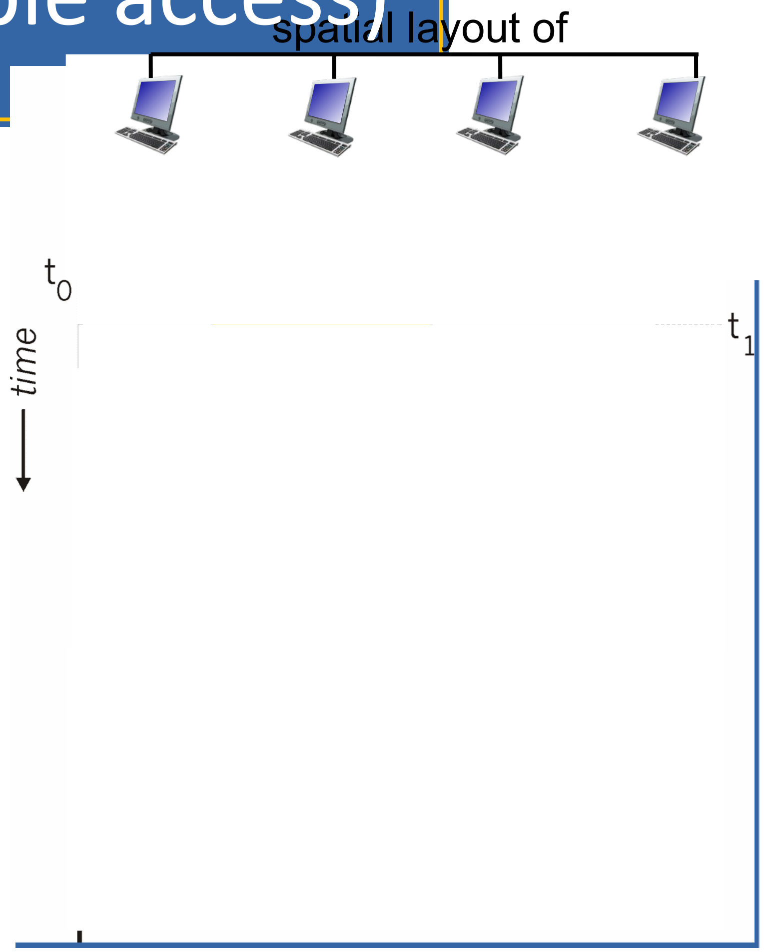
simple CSMA: listen before transmit:

- if channel sensed idle: transmit entire frame
- if channel sensed busy: defer transmission

- human analogy: don't interrupt others!

CSMA (carrier sense multiple access)

- collisions *can* still occur with carrier sensing:
  - propagation delay means two nodes may not hear each other's just-started transmission
- collision: entire packet transmission time wasted
  - distance & propagation delay play role in in determining collision probability



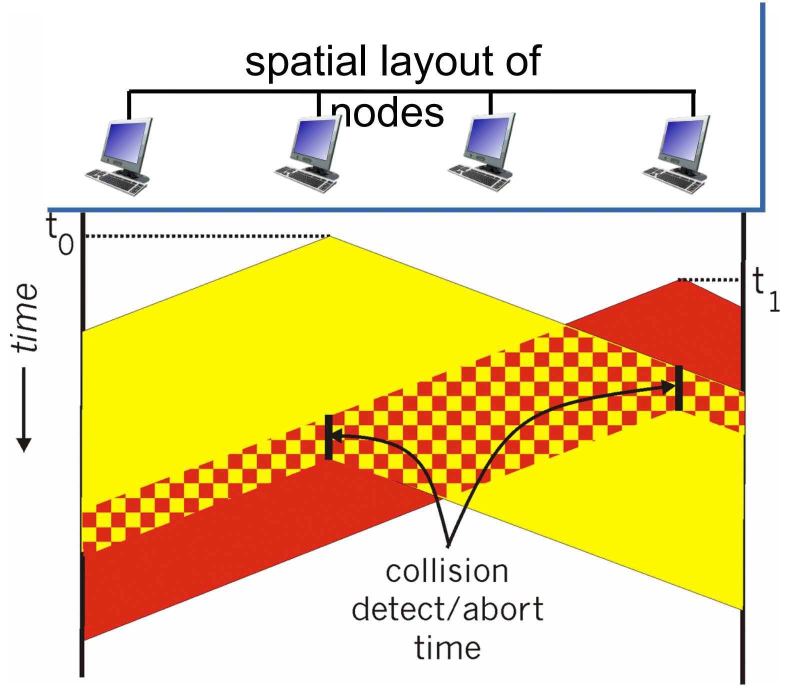
CSMA/CD

CSMA/CD: CSMA with *collision detection*

- collisions *detected* within short time
- colliding transmissions aborted, reducing channel wastage
- collision detection easy in wired, difficult with wireless
- human analogy: the polite conversationalist

CSMA/CD

- CSMA/CS reduces the amount of time wasted in collisions
  - transmission aborted on collision detection



Ethernet CSMA/CD

1. NIC receives datagram from network layer, creates frame
2. If NIC senses channel:
   if idle: start frame transmission.
   if busy: wait until channel idle, then transmit
3. If NIC transmits entire frame without collision, NIC is done with frame !
4. If NIC detects another transmission while sending: abort, send jam signal
5. After aborting, NIC enters *binary (exponential) backoff:*
   - after *m*th collision, NIC chooses *K* at random from $\{0,1,2, \ldots, 2^m-1\}$. NIC waits $K \cdot 512$ bit times, returns to Step 2
   - more collisions: longer backoff interval

Link Layer: 6-34

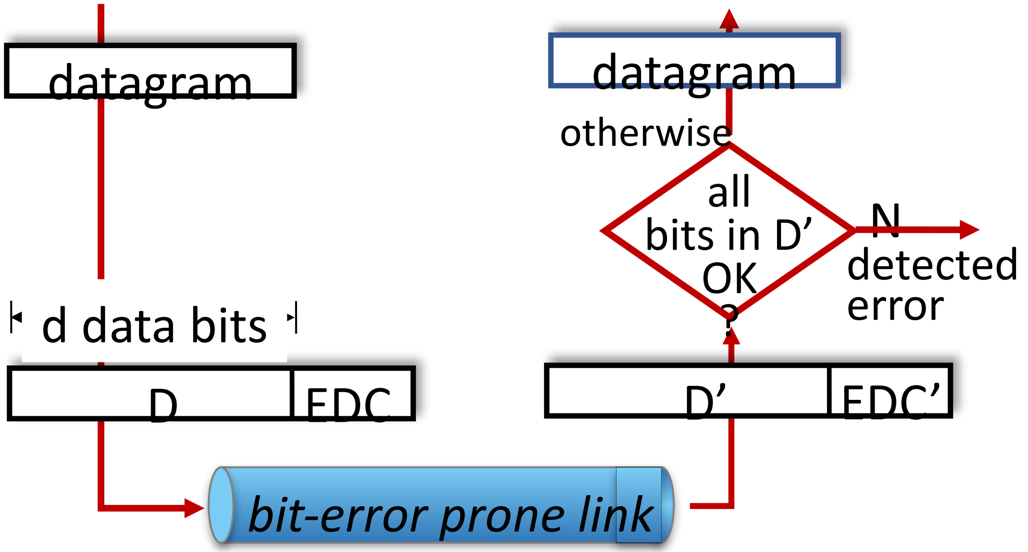
Error Detection

- Parity
- Checksumming
- CRC (Cyclic Redundancy Check)

Error Detection

EDC: error detection and correction bits (e.g., redundancy)

D: data protected by error checking, may include header fields



Error detection not 100% reliable!

- protocol may miss some errors, but rarely
- larger EDC field yields better detection and correction

Parity

single bit parity:

- detect single bit errors

0111000110101011	1

d data bits | parity bit

Even parity: set parity bit so there is an even number of 1's

two-dimensional bit parity:

- detect *and correct* single bit errors

row parity →

column parity ↓

$$\begin{array}{cccc|c} d_{1,1} & \cdots & d_{1,j} & & d_{1,j+1} \\ d_{2,1} & \cdots & d_{2,j} & & d_{2,j+1} \\ \cdots & \cdots & \cdots & & \cdots \\ d_{i,1} & \cdots & d_{i,j} & & d_{i,j+1} \\ \hline d_{i+1,1} & \cdots & d_{i+1,j} & & d_{i+1,j+1} \end{array}$$

no errors:

$$\begin{array}{ccccc|c} 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ \hline 1 & 0 & 1 & 0 & 1 & 0 \end{array}$$

detected and correctable single-bit error:

$$\begin{array}{ccccc|c} 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ \hline 1 & 0 & 1 & 0 & 1 & 0 \end{array}$$

parity error

parity error

Internet checksum

Goal: detect errors (*i.e.,* flipped bits) in transmitted segment

sender:

- treat contents of segment as sequence of 16-bit integers
- checksum: addition (one's complement sum) of segment content
- checksum value put into checksum field

receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - not equal - error detected
  - equal - no error detected. *But maybe errors nonetheless?* More later ....

Internet checksum

Goal: detect errors (*i.e.,* flipped bits) in transmitted segment

	1st number	2nd number	sum
Transmitted:	5	6	11
Received:	4	6	11

receiver-computed checksum ≠ sender-computed checksum (as received)

Internet checksum

Goal: detect errors (*i.e.,* flipped bits) in transmitted segment

sender:

- treat contents of UDP segment (including UDP header fields and IP addresses) as sequence of 16-bit integers
- checksum: addition (one's complement sum) of segment content
- checksum value put into UDP checksum field

receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - not equal - error detected
  - equal - no error detected. *But maybe errors nonetheless?* More later ….

Internet checksum

example: add two 16-bit integers

```
            1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
            1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
            -------------------------------
wraparound 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
            -------------------------------
       sum  1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0
  checksum  0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1
```

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

Internet checksum

example: add two 16-bit integers

```
         1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0    0 1
         1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1    1 0
         -------------------------------
       1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
         -------------------------------
sum      1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0
checksum 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1
```

Even though numbers have changed (bit flips), *no* change in checksum!

Cyclic Redundancy Check (CRC)

- more powerful error-detection coding
- D: data bits (given, think of these as a binary number)
- G: bit pattern (generator), of *r+1* bits (given)

r CRC bits

← d data bits → ← →

D	R

bit pattern

$$<D,R> = D * 2^r \ \text{XOR} \ R$$

formula for bit pattern

goal: choose *r* CRC bits, R, such that <D,R> exactly divisible by G (mod 2)

- receiver knows G, divides <D,R> by G. If non-zero remainder: error detected!
- can detect all burst errors less than r+1 bits
- widely used in practice (Ethernet, 802.11 WiFi)

CRC

We want:

$$D \cdot 2^r \ \ XOR \ \ R = nG$$

or equivalently:

$$D \cdot 2^r = nG \ \ XOR \ \ R$$

or equivalently:

if we divide $D \cdot 2^r$ by G, want remainder R to satisfy:

$$R = remainder\left[\frac{D \cdot 2^r}{G}\right]$$

G

```
                 1 0 1 0 1 1
         _____________________
1 0 0 1 ) 1 0 1 1 1 0 0 0 0
          1 0 0 1
              1 0 1
              0 0 0
              1 0 1 0
              1 0 0 1
                  1 1 0
                  0 0 0
                  1 1 0 0
                  1 0 0 1
                    1 0 1 0
                    1 0 0 1
```

D * 2^r