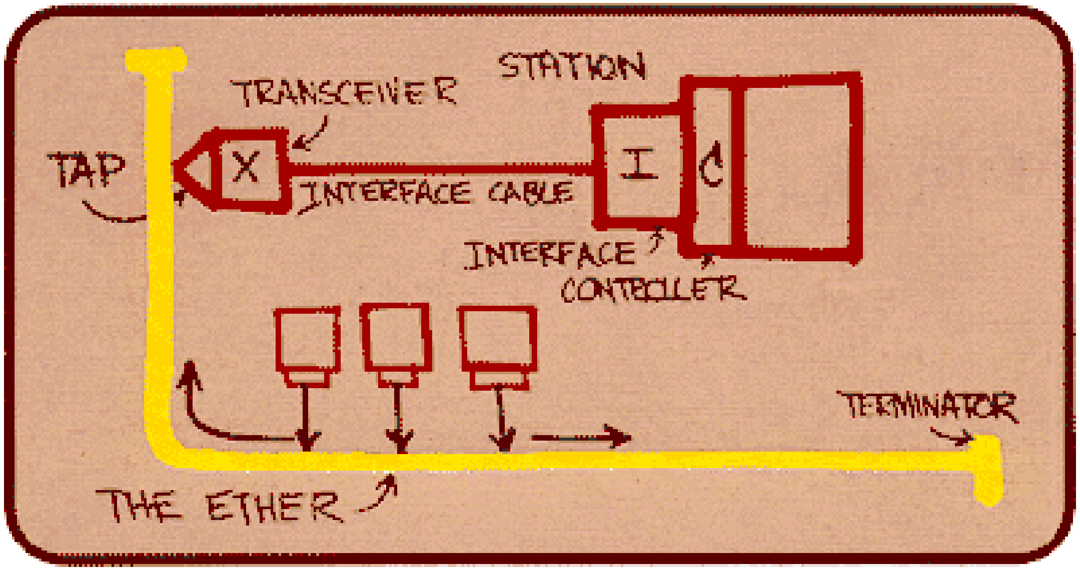
Randomization

- Randomization used in many protocols
- We'll study examples:
  - Ethernet multiple access protocol
  - Reliable multicast
  - Router (de)synchronization
  - Switch scheduling
  - Active queue management

Ethernet

- Single shared broadcast channel
- 2+ simultaneous transmissions by nodes: interference
  - only one node can send successfully at a time
- *Multiple access protocol:* Distributed algorithm that determines how nodes share channel, i.e., determine when node can transmit



Metcalfe's Ethernet sketch

Deterministic algorithms

- Time Division Multiplexing ?

- Polling?

- Virtual Ring?

Ethernet: uses CSMA/CD

A: sense channel, **if** idle

```
then {
        transmit and monitor the channel;
        If detect another transmission
         then {
              abort and send jam signal;
              update # collisions;
              delay as required by exponential backoff algorithm;
              goto A
              }
        else {done with the frame; set collisions to zero}
     }
else {wait until ongoing transmission is over and goto A}
```

Ethernet's CSMA/CD (more)

Jam Signal: Make sure all other transmitters are aware of collision; 48 bits;

Exponential Backoff:

- First collision for given packet: choose K randomly from $\{0,1\}$; delay is $K \text{ x } 512$ bit transmission times
- After second collision: choose K randomly from $\{0,1,2,3\}$ ...
- After ten or more collisions, choose K randomly from $\{0,1,2,3,4,\ldots,1023\}$

Ethernet's use of randomization

- *Resulting behavior:* Probability of retransmission attempt (equivalently length of randomization interval) adapted to current load
  - simple, load-adaptive, multiple access

more collisions → heavier Load (most likely), more nodes trying to send → randomize retransmissions over longer time interval, to reduce collision probability

Ethernet comments

- Upper bounding at $1023 = k$ limits max size
- Could remember last value of K when we were successfull (analogy: TCP remembers last values of congestion window size)
- Q: Why use binary backoff rather than something more sophisticated such as AIMD in TCP congestion control: simplicity
  - Note: Ethernet does multiplicative-increase-complete-decrease

The bottom line

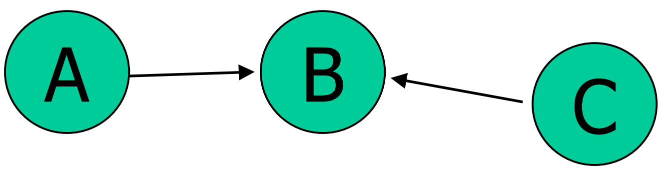
- Why does Ethernet use randomization:
  To desynchronize:

  A distributed adaptive algorithm to spread out load over time when there is contention for multiple access channel.

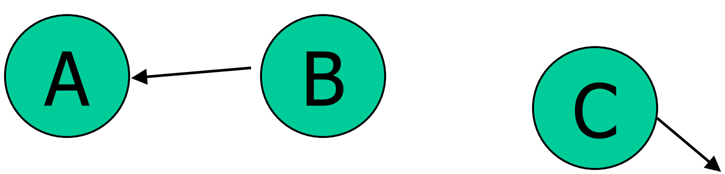
Excursion: What if wireless?!

Typical wireless networks...:

- are not full-duplex (just one channel)
- nodes cannot sense the medium during own transmissions (just one antenna...)
- no bounded propagation domain
- are multihop (hidden and exposed terminal problems):



Hidden terminal: C does not notice that B is currently receiving transmissions from A also => no „remote carrier sense"



Exposed terminal: B sends A and C wants to send to someone on the right: it waits because it hears B, but B would not reach the recipient of C, so actually C could send! => inefficient

Excursion: Wireless MAC?

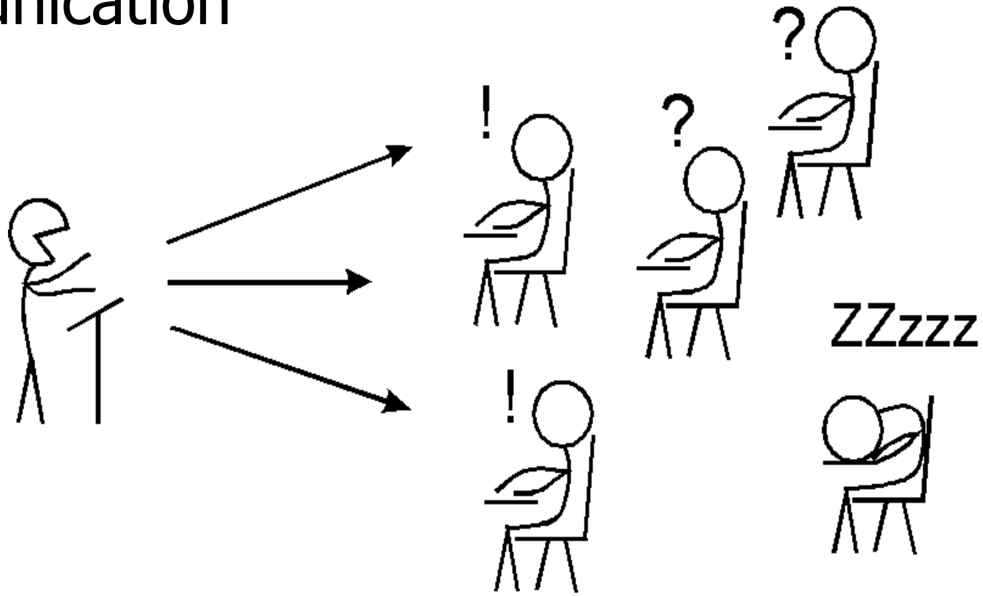
Therefore, CD is often replaced by (best effort) Collision Avoidance (CA, cf *DIFS/SIFS* etc.)

Still ongoing research, e.g., there are randomized distributed medium access protocols which optimally coordinate medium access probabilities and exploit the unpredictable non-jammed (e.g., due to external inteference) time periods (e.g., the Jade protocol).

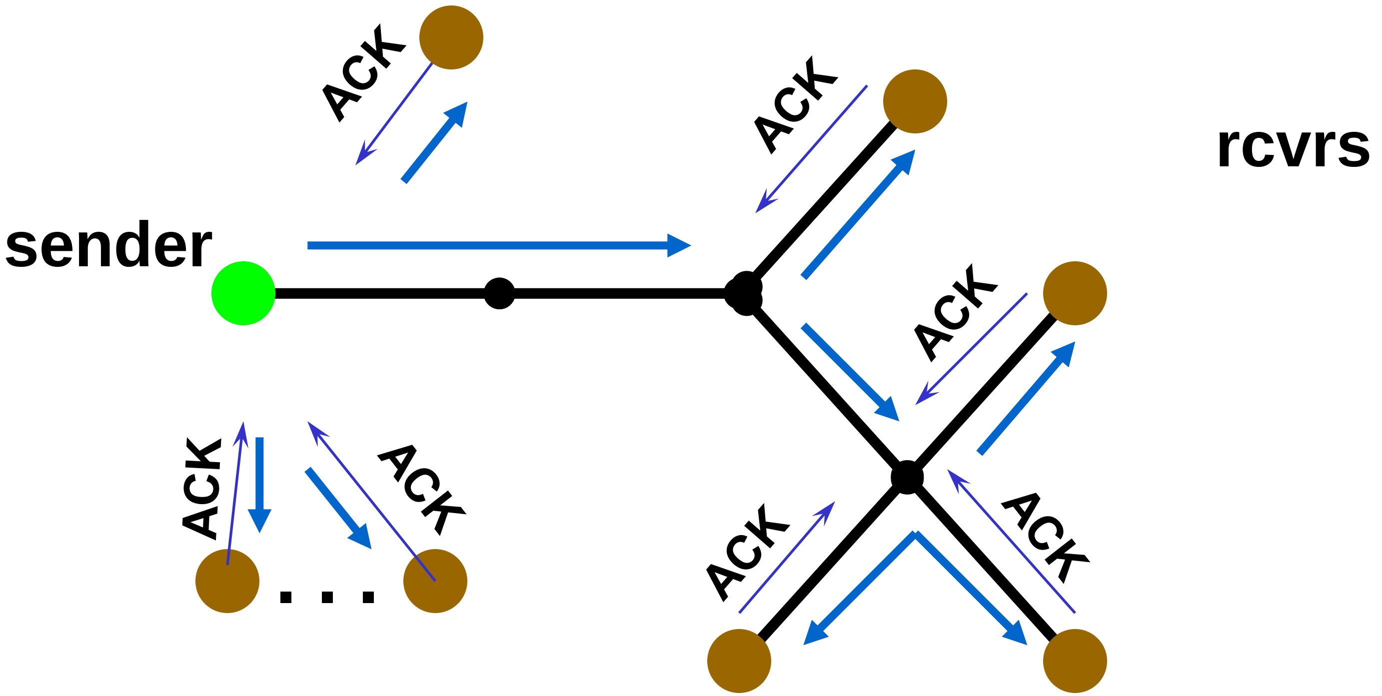
A Jamming-Resistant MAC Protocol for Multi-Hop Wireless Networks, DISC 2010.

Randomization in Reliable Multicast

- ***RM:*** How to transfer data "reliably" from source(s) to R receivers.
- *Conjecture:* All current RM error and congestion control approaches have an analogy in human-human communication



Scalability: Feedback Implosion



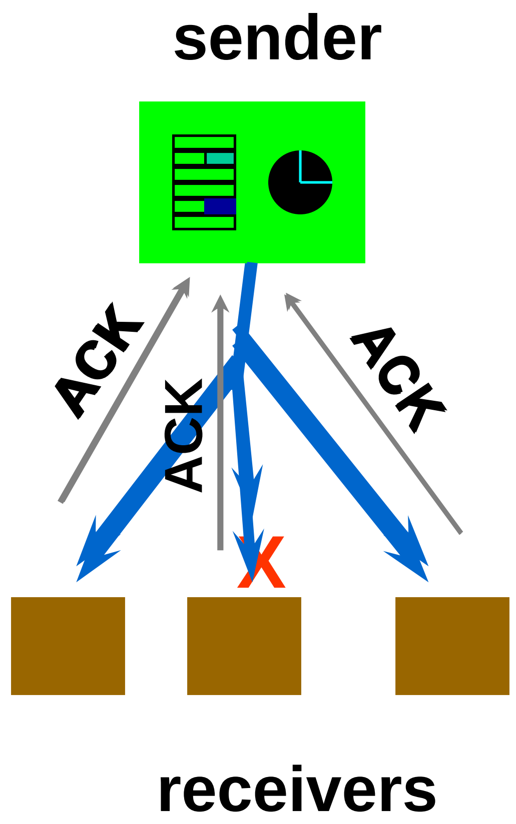
Sender Oriented Reliable Mcast

Sender:

- mcasts all (re)transmissions
- selective repeat
- timers for loss detection
- ACK table
- pkt removed when *all* ACKs are in

Rcvr: ACKs received pkts

Note: Group membership important



(Simple) Rcvr Oriented Reliable Mcast

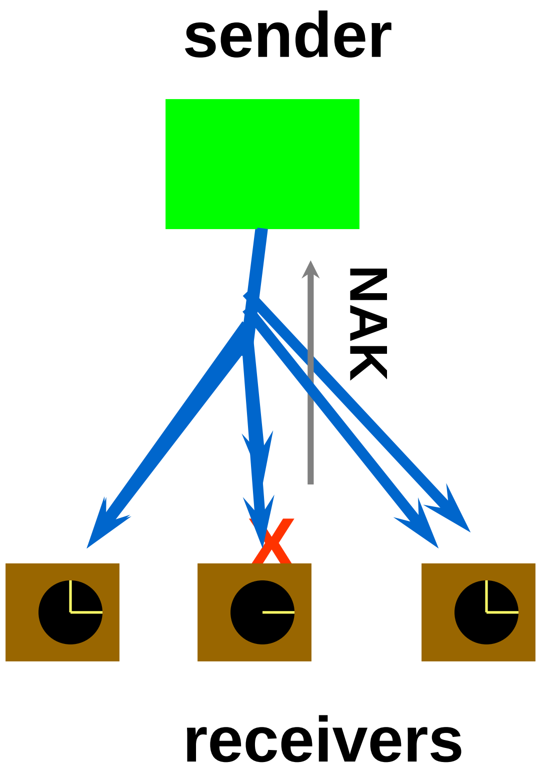
Sender:

- mcasts (re)transmissions
- selective repeat
- responds to NAKs
- when no longer buffer pkt?

Rcvr:

- NAKs (unicast to sender) missing pkts
- timer to detect lost retransmission

Note: easy to allow joins/leaves



Receiver- versus sender-oriented RM: observations

Rcvr-oriented: Shift recovery burden to rcvrs

- Loss detection "responsibility", timers
- Scaling: computational power grow as R grows
- Weaker notion of "group"
- Receivers can transparently choose different reliability semantics

But ……

- When does sender "release" data rcvd by all?
- Heartbeat needed to detect lost last pkt

RM: Coping with Scale, Heterogeneity

Issues:

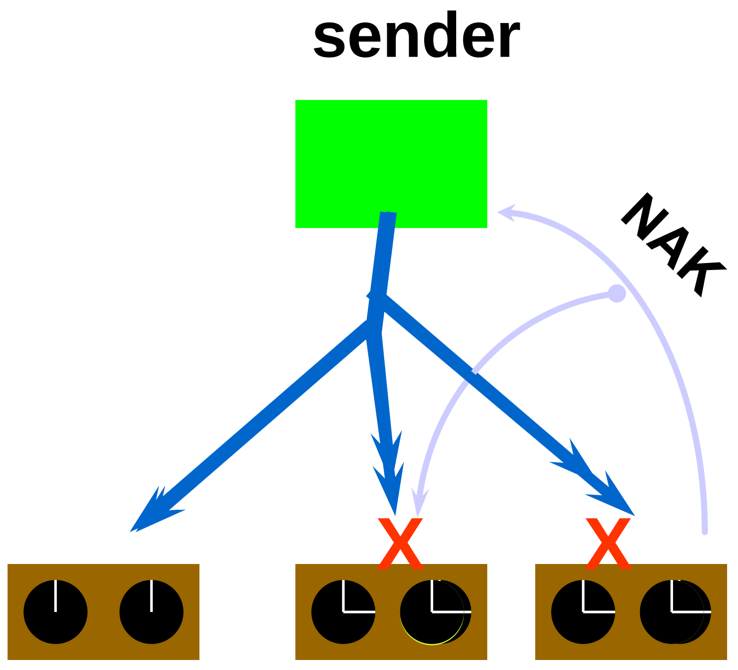
- Avoid feedback implosion in reverse path
- Avoid receiving unneeded data (retrans.) in forward path
- Recover data quickly, avoid long repair times

Techniques:

- Feedback suppression
- Local recovery

Feedback suppression

- Randomly delay NAKs
  - "Listen" to NAKs generated by others
  - If no NAK for lost pkt when timer expires, multicast NAK
- Widely used in RM
- Tradeoffs
  - Reduces bandwidth
  - Additional complexity at receivers (timers, etc)



Reliable multicast (SRM)

Use of randomization

- Avoid synchronizing all replies
- To reduce feedback implosion
- In local recovery, to reduce number of retransmissions of same message.
- Could scale the randomization interval to be load-adaptive.