

Part 2: Design Principles

Goals:

- identify, study common architectural principles, protocol mechanisms
- *synthesis*: big picture

Overview:

- Separation of data and control: signaling
- state management: hard vs. soft-state
- randomization
- indirection
- multiplexing
- virtualization
- design for scale

Part 2 2-1

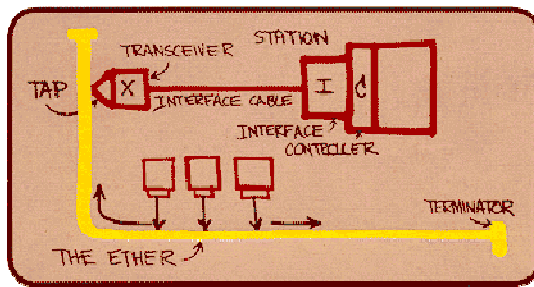
3. Randomization

- randomization used in many protocols
- we'll study examples:
 - Ethernet multiple access protocol
 - queue management
 - reliable multicast
 - router (de)synchronization

Part 2 2-2

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

Part 2 2-3

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}
```

Part 2 2-4

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 $K \times 512$ bit transmission times
- after second collision: choose K randomly from {0,1,2,3}...
- after next collision double the range for K (and keep doubling on collisions until.....)
- after ten or more collisions, choose K randomly from {0,1,2,3,4,...,1023}

Part 2 2-5

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

Part 2 2-6

Ethernet comments

- upper bounding at $1023 = k$ limits max size
- could remember last value of K when we were successful (analogy: TCP remembers last values of congestion window size)
- Q: why use binary back-off rather than something more sophisticated such as AIMD: simplicity
 - note: Ethernet does multiplicative-increase-complete-decrease (why?)

Part 2 2-7

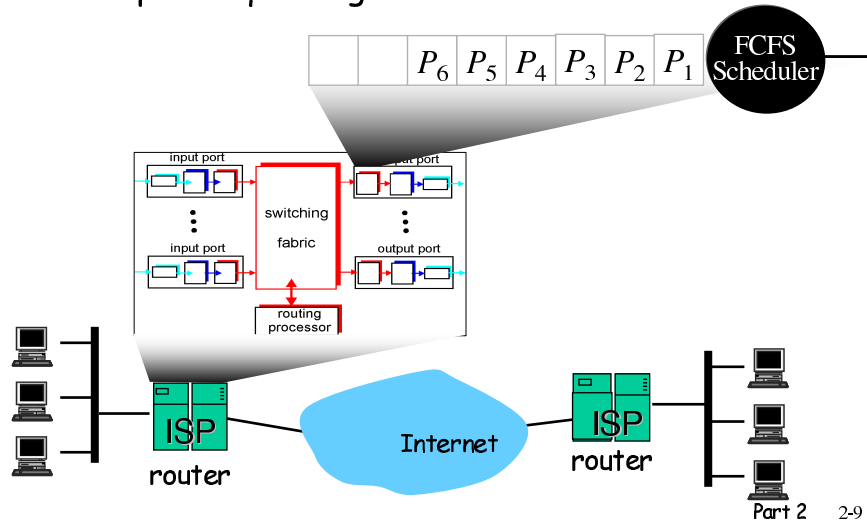
The bottom line

- 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

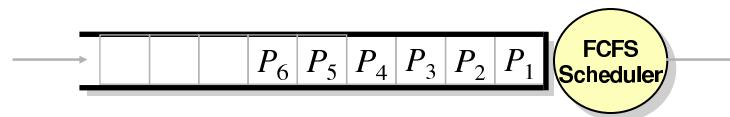
Part 2 2-8

Randomization in Router Queue Management

- normally, packets dropped only when queue overflows
 - "Drop-tail" queuing



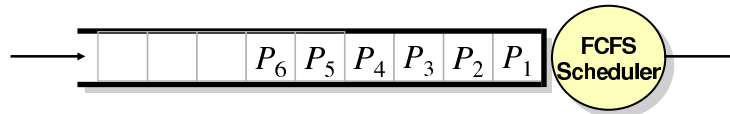
The case against drop-tail queue management



- large queues in routers are "a bad thing"
 - End-to-end latency dominated by length of queues at switches in network
- allowing queues to overflow is "a bad thing"
 - connections transmitting at high rates can starve connections transmitting at low rates
 - connections can *synchronize* their response to congestion (why?)

Part 2 2-10

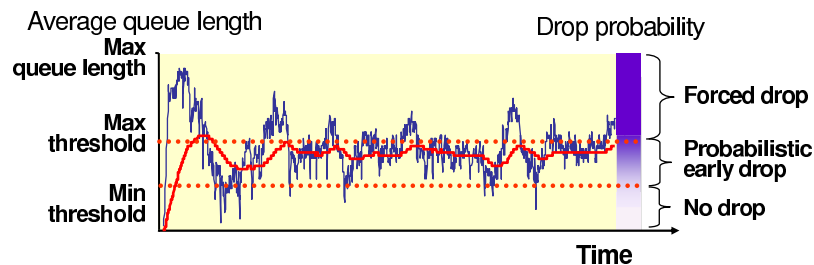
Idea: early random packet drop



- When queue length exceeds threshold, packets dropped with fixed *probability*
 - probabilistic packet drop: flows see same loss rate
 - problem: bursty traffic (burst arrives when queue is near full) can be over-penalized

Part 2 2-11

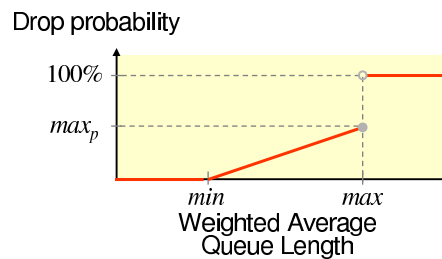
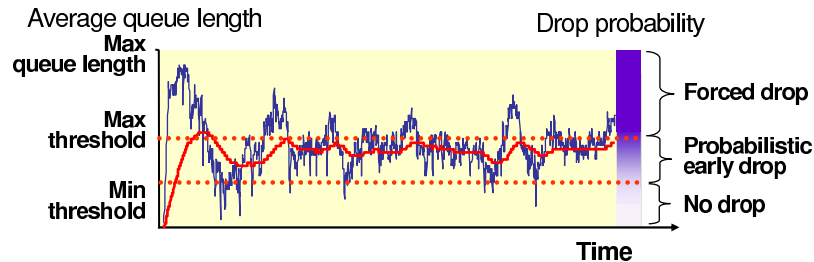
Random early detection (RED) packet drop



- use exponential *average* of queue length to determine when to drop
 - avoid overly penalizing short-term bursts
 - react to longer term trends
- tie drop prob. to weighted avg. queue length
 - avoids over-reaction to mild overload conditions

Part 2 2-12

Random early detection (RED) packet drop



Part 2 2-13

RED Algorithm

```

avg = 0; count = -1;
For each packet arrival {
  Average queue size:
    if queue not empty:  $avg = (1-w_q) * avg + w_q * q$ 
    else  $m = f(\text{time} - q_{\text{time}})$ ;  $avg = (1-w_q)^m * avg$ ;
  if  $min_{th} \leq avg < max_{th}$  {
    count++;
     $p_b = max_p * (avg - min_{th}) / (max_{th} - min_{th})$ ;
     $p_a = p_b / (1 - count * p_b)$ ;
    Mark packet with probability  $p_a$ ; if marked, count = 0; }
  else if  $max_{th} \leq avg$  {
    Mark packet; count = 0}
  else count = -1;
}

```

Part 2 2-14

RED: why probabilistic drop?

- ❑ provide gentle transition from no-drop to all-drop
 - provide "gentle" early warning
- ❑ provide same loss rate to all sessions:
 - with tail-drop, low-sending-rate sessions can be completely starved
- ❑ avoid synchronized loss bursts among sources
 - avoid cycles of large-loss followed by no-transmission

Part 2 2-15

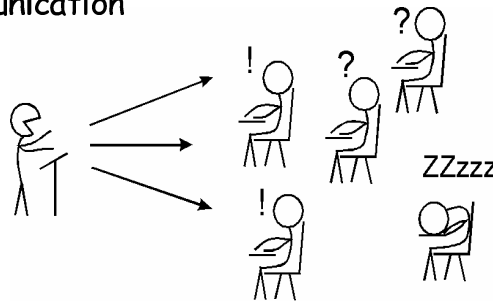
Random Early Detection (RED): Comments

- ❑ large number (5) of parameters: difficult to tune (at least for http traffic)
- ❑ gains over drop-tail FCFS not that significant
- ❑ still not widely deployed ...

Part 2 2-16

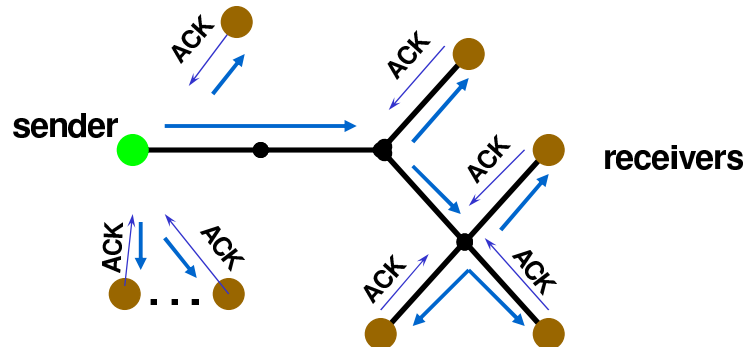
Randomization in Reliable Multicast

- **RM:** how to transfer data "reliably" from source(s) to R receivers; $R = 10 \text{ -- } 100 \text{ -- } 1000 \text{ -- } 10000 \text{ -- } 100000$
- **conjecture:** all current RM error and congestion control approaches have an analogy in human-human communication



Part 2 2-17

Scalability: Feedback Implosion



Part 2 2-18

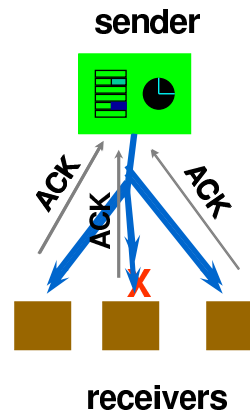
Sender Oriented Reliable Multicast

Sender:

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

Rcvr: ACKs received pkts

Note: group membership important



Part 2 2-19

(simple) Receiver-oriented Reliable Multicast

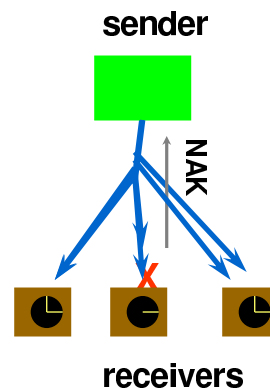
Sender:

- mcasts (re)transmissions
- selective repeat
- responds to NAKs
- when can the sender discard packet?

Rcvr:

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

Note: easy to allow joins/leaves



Part 2 2-20

Receiver- versus sender-oriented RM: observations

Rcvr-oriented: shift recovery burden to receivers

- loss detection "responsibility", timers
- scaling: protocol computational resources grow as R grows
- weaker notion of "group"
- receivers can transparently choose different reliability semantics

but

- when does sender "release" buffered packets?
- heartbeat needed to detect lost last packet

Part 2 2-21

RM: Coping with Scale, Heterogeneity

Issues:

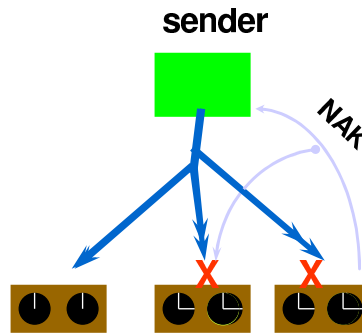
- avoid feedback implosion in reverse path
- avoid receiving unneeded data (retrans.) in forward path
- recover data quickly, avoid long repair times

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 (recall similar IGMP idea)
- tradeoffs
 - reduces bandwidth
 - additional complexity at receivers (timers, etc)



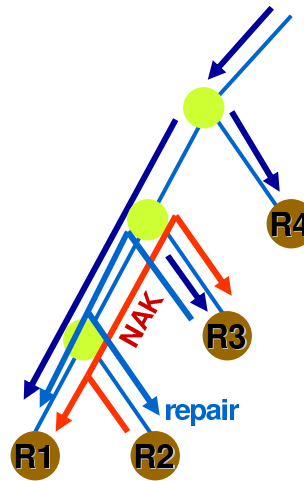
Part 2 2-23

Local Recovery in SRM

- Allow rcvr to recover lost pkt from "nearby" rcvr
 - "ask your neighbor": send localized NAK (repair request)
 - multicast: **randomize** local repair transmission time to avoid too many replies
- orthogonal (complementary) to feedback suppression
- who to recover from?
 - don't want repair request to go to everyone
 - scoping: how to restrict how far request will travel: IP time-to-live field

Local Recovery: example

- R2 detects lost pkt
- multicasts repair request
- limited scope
 - not seen by R4
- R1 and R3 have pkt
 - R3 times out first and sends repair



Part 2 2-25

Reliable multicast (SRM)

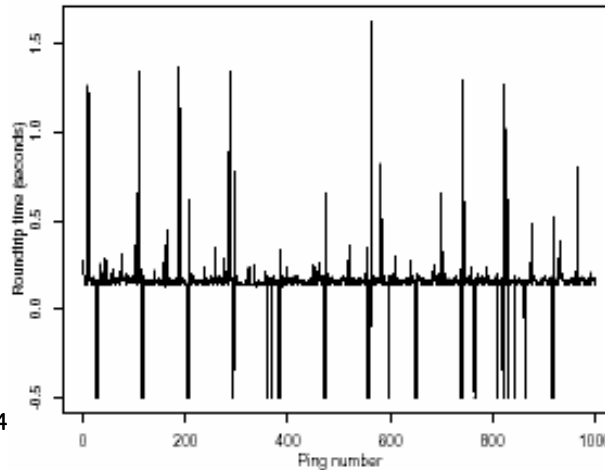
Use of randomization

- used in repairing losses, so reduced chance of multiple receivers sending (NAK, repairs) at same time

Part 2 2-26

(de)Synchronization of periodic routing updates

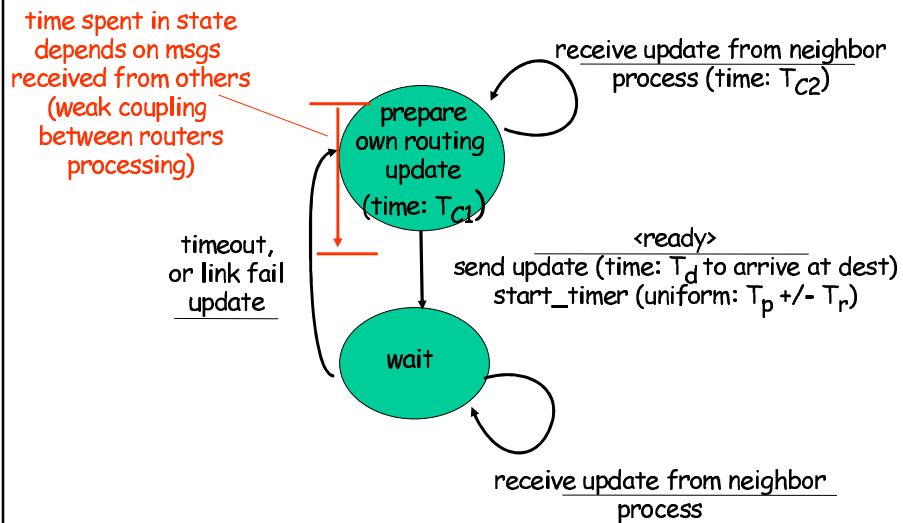
- periodic losses observed in end-end Internet traffic
- Why?



source: Floyd,
Jacobson 1994

part 2 2-27

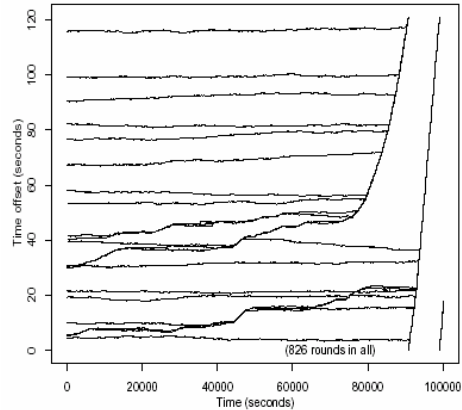
Router update operation:



Part 2 2-28

Router synchronization

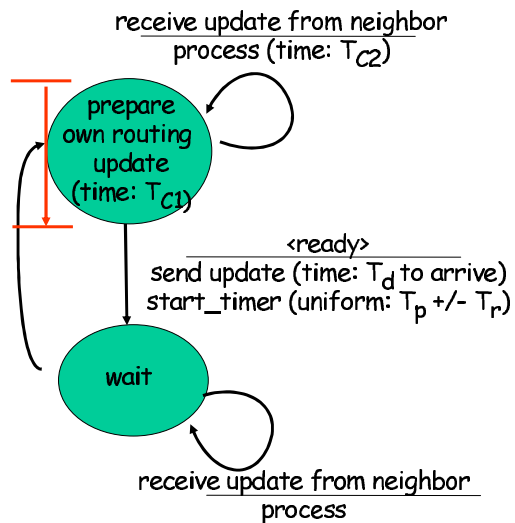
- 20 (simulated) routers broadcasting updates to each other
- x-axis: time until routing update sent relative to start of round
- By $t=100,000$ all router rounds are of length 120!
- synchronization or lack thereof depends on system parameters



Part 2 2-29

Avoiding synchronization

- choose random timer component, T_r large (e.g., several multiples of T_{c1})



Part 2 2-30