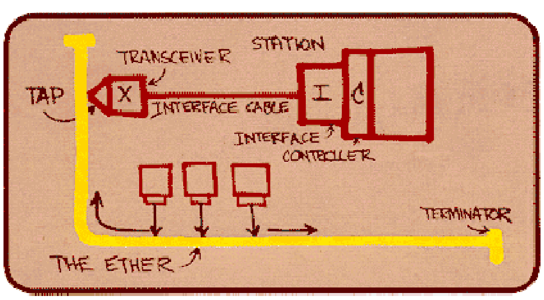
Randomization

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

1

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

2

Deterministic algorithms

- Time Division Multiplexing ?
- Polling?
- Virtual Ring?

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}
```

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 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 $\{0,1,2,3,4,\ldots,1023\}$

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

6

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: simplicity
  - Note: Ethernet does multiplicative-increase-complete-decrease (why)

7

The bottom line

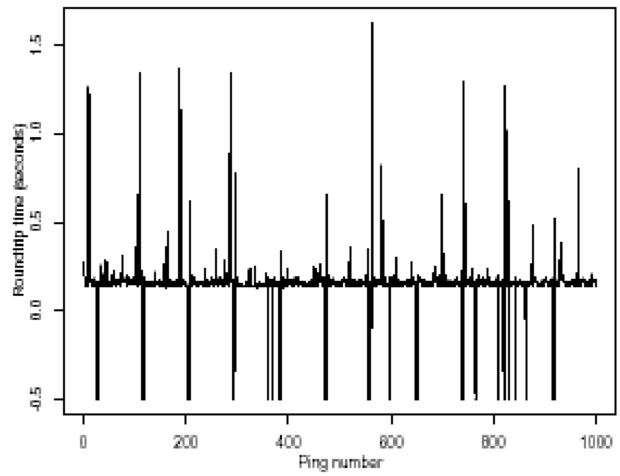
- Why does Ethernet use randomization: to desynchronize:

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

8

(de)Synchronization of periodic routing updates

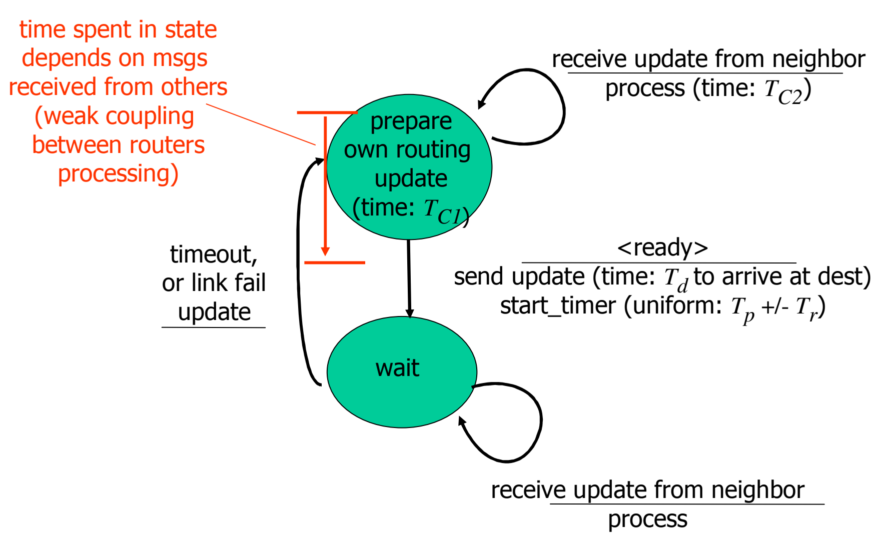
- Periodic losses observed in end-end Internet traffic
- Why?



source: Floyd, Jacobson 1994

9

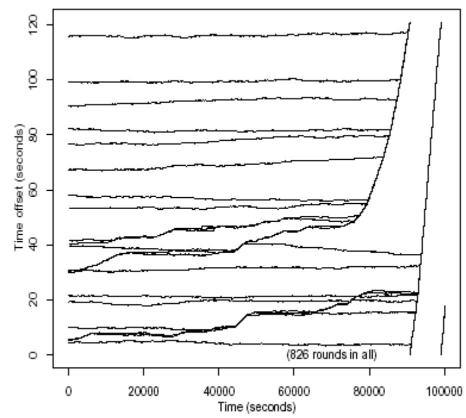
Router update operation:



10

Router synchronization

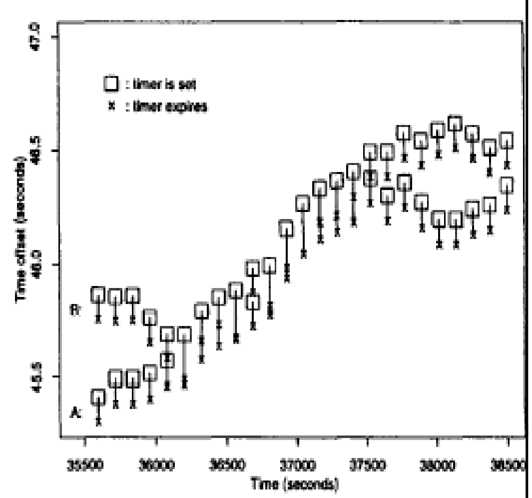
- 20 routers broadcasting updates to each other
- x-axis: Time until routing update sent relative to start of round
- By t=100,000 *all* router rounds are of length 120!
- Synchronization or lack thereof depends on system parameters



11

Zooming in ...

- Blowup of previous graph
- Note expansion of computation phase
  → increased
  period



12

Sync

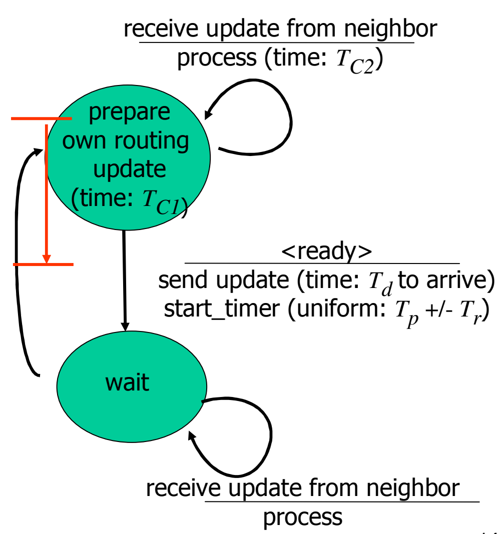
- Coupled routers
- Example of spontaneous synchronization
  - Fireflies
  - Sleep cycle
  - Heart beat
  - etc.

    Steven Strogatz . *Sync*, Hyperion Books, 2003.

13

Avoiding synchronization

- Enforce max time spent in prepare state
- Choose random timer component, T_r large



14

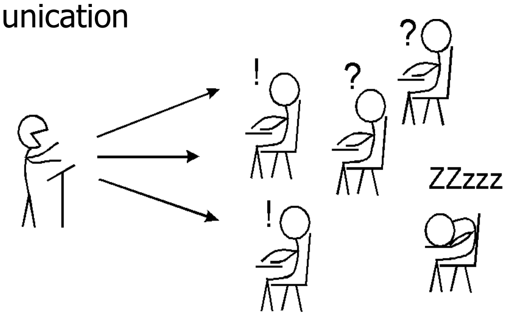
Router (de)synchronization

Use of randomization: Desynchronize routers!

15

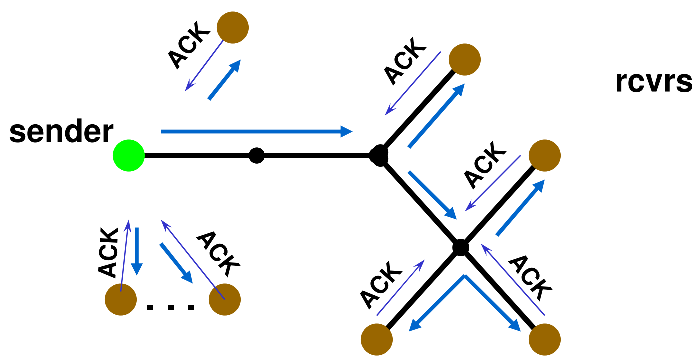
Randomization in Reliable Multicast

- ***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



16

Scalability: Feedback Implosion



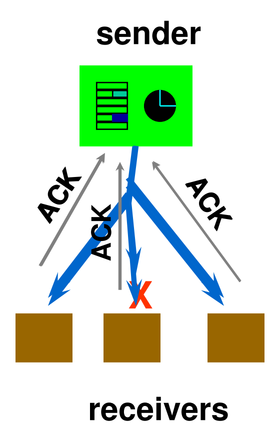
17

Sender Oriented Reliable Mcast

Sender:
- mcasts 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



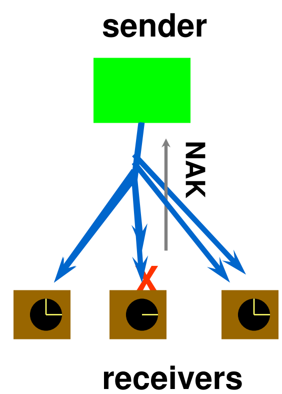
18

(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



19

Receiver- versus sender-oriented RM: observations

Rcvr-oriented: Shift recovery burden to rcvrs
- 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" data rcvd by all?
- Heartbeat needed to detect lost last pkt

20

Evaluation of Approaches

Examine resource requirements

- processing requirements
  - expected time to process pkt
    - at sender: $X, E[X]$
    - at rcvr: $Y, E[Y]$
  - mean value approach
- network requirements

Assumptions for Analysis

- one sender, R receivers
- independent errors, p per rcvr
- lossless signaling

M - total number of transmissions per packet

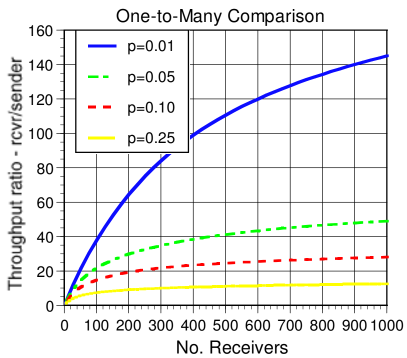
$$P[M \le m] = \left(1 - p^m\right)^R, \qquad m = 1, \ldots$$

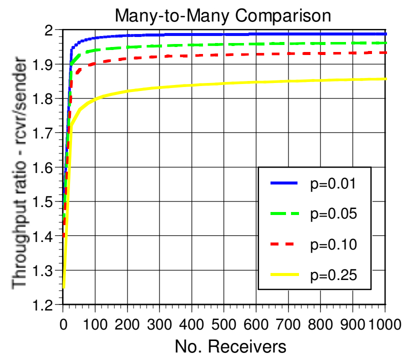
$$E[M] = \sum_{m=1}^{\infty} 1 - \left(1 - p^m\right)^R$$

- ...

Sender vs. Receiver (cont.)

Metric - rcvr oriented thruput/sender oriented thruput





Significant performance improvement shifting burden to receivers for 1-many; not as great for many-many

23

RM: Coping with Scale, Heterogeity

Issues:

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

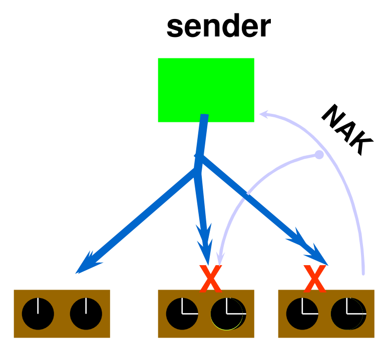
Techniques:

- feedback suppression
- local recovery

24

Feedback Suppression

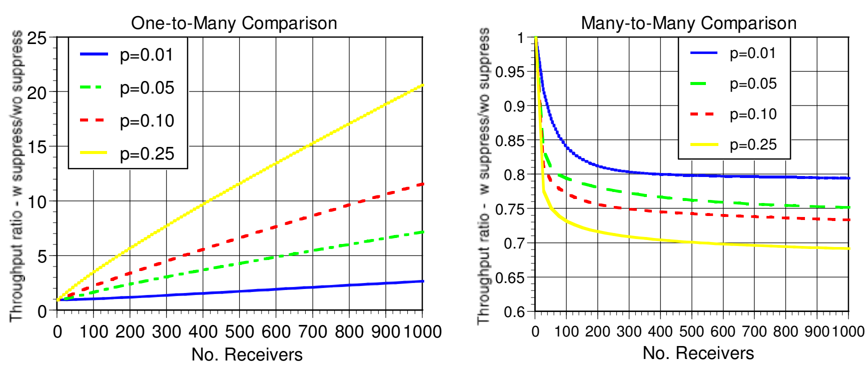
- Randomly delay NAKs
  - "Listen" to NAKs generated by others
  - If no NAK for lost pkt when timer expires, multicast NAK
- Widely used in RM
- Tradeoffs
  - Reduces bandwidth
  - Additional complexity at receivers (timers, etc)



25

Feedback Suppression: Performance gains

Metric - suppression thruput/no suppression thruput



gains/loss depends on whether 1-many or many-many

26

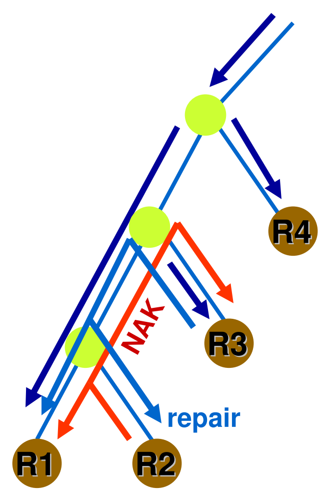
Local Recovery in SRM

- 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

27

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



28

Reliable multicast (SRM)

Use of randomization

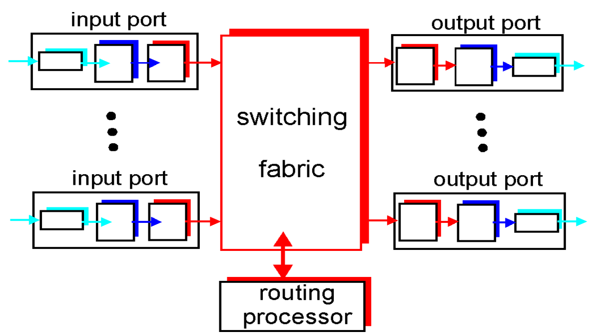
- Avoid synchronizing all replies
- To reduce feedback implosion
- In local recovery, to reduce number of retransmissions of same message.
- Could scale the randomization interval to be load-adaptive.

29

Randomization in Switch Scheduling

Two key router functions:

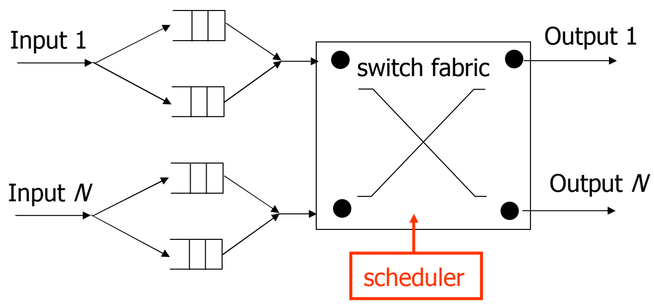
- Run routing algorithms/protocol (RIP, OSPF, BGP)
- *Switching* datagrams from incoming to outgoing link (our focus here)



30

Input queueing with bufferless outputs

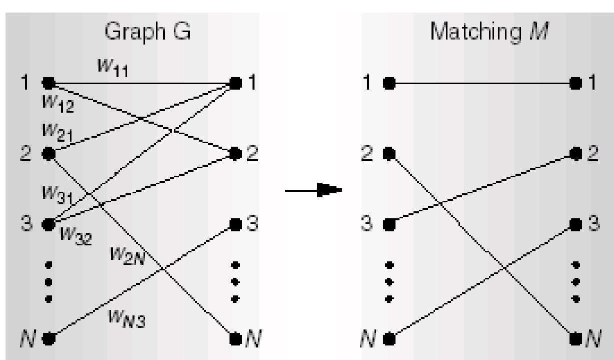
- N input ports, output ports
- Each port has N per-output port queues
- Matching problem: In each time slot
  - Each output port can receive at most one packet from among $N \times N$ input queues
  - Each input port (N queues) can forward at most one packet



31

Solving the assignment problem

- W_{ij}: Queue length (weight) of input i, queue j
- Maximum weight matching solution: $O(N^3)$ that can deliver 100% throughput
- Can we do better?

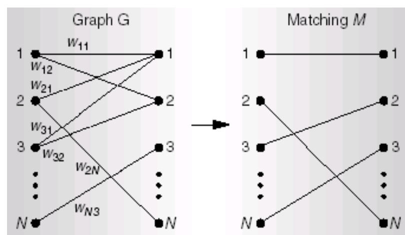


32

Using randomization to optimize

Key observations:

1. State of switch (input queue lengths) changes little from slot to slot
   - "good" solution at t, is close to "good" solution at $t+1$
2. A new, randomly-chosen solution will sometimes be better solution at $t+1$ than solution used at t

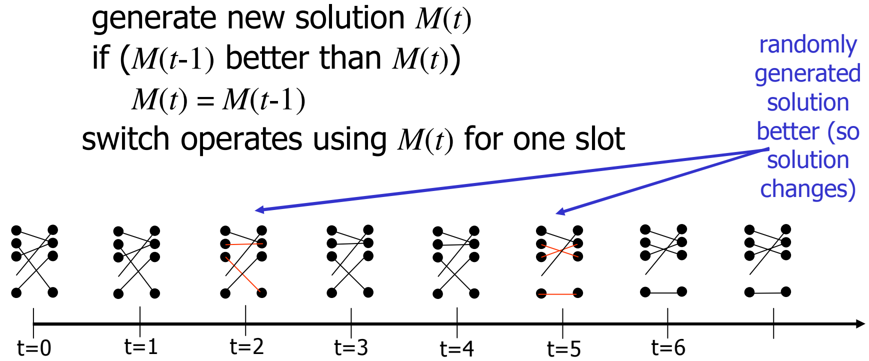


33

Using randomization to optimize

Algorithm:

$t=0$ choose any matching $M(0)$
switch operates using $M(0)$ for one slot
for ($t=1$;   ; t++)
  generate new solution $M(t)$
  if ($M(t-1)$ better than $M(t)$)
    $M(t) = M(t-1)$
  switch operates using $M(t)$ for one slot



34

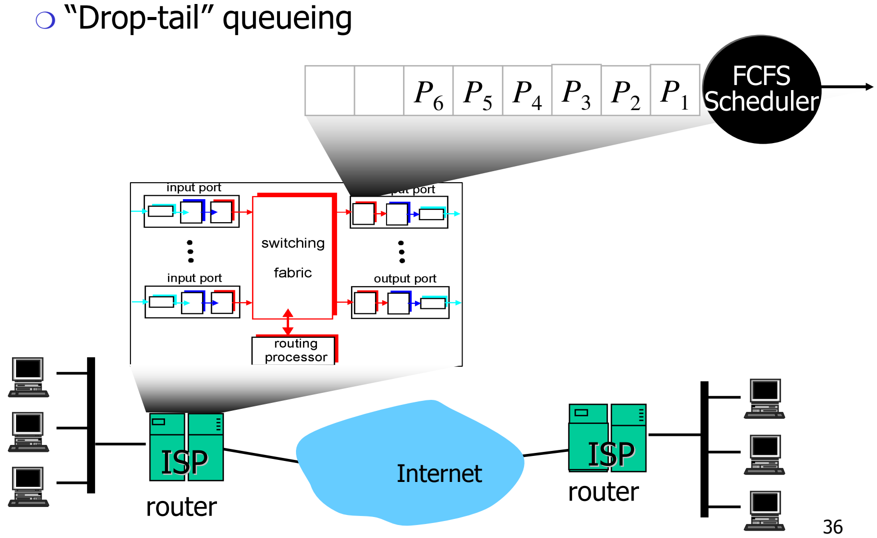
Using randomization to optimize

Result [Tassiulas]: Randomized algorithm achieves 100% thoughput (i.e., as good as brute force, non-randomized $O(N^3)$ optimization algorithm).

35

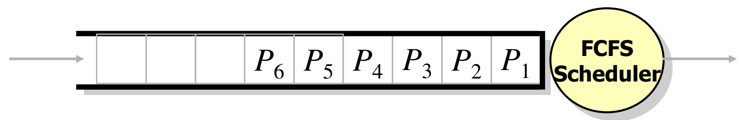
Randomization in Router Queue Management

- Normally, packets dropped only when queue overflows
  - "Drop-tail" queueing



36

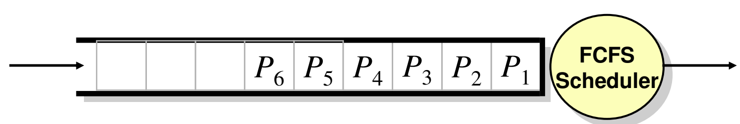
The case against drop-tail queue management



- 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

37

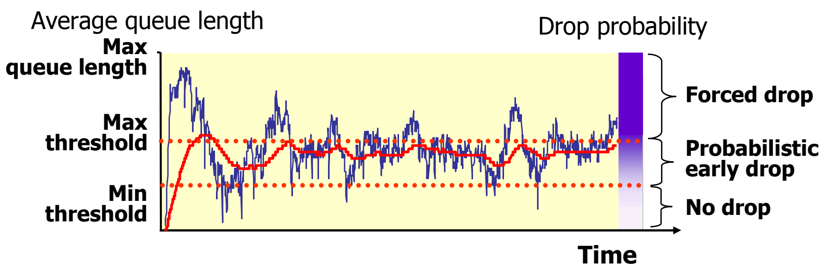
Idea: Early random packet drop



- 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 overpenalized

38

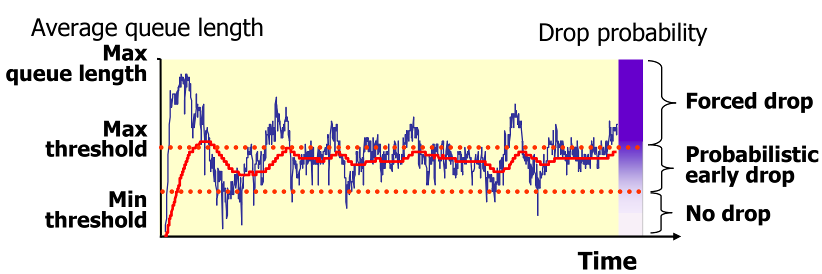
Random early detection (RED) packet drop

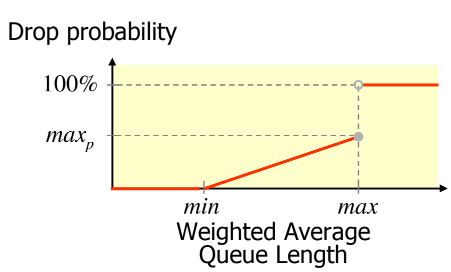


- Use exponential *average* of queue length to determine when to drop
  - Avoid overly penalizing short-term bursts
  - React to longer term trends
- Tie drop prob. to weighted avg. queue length
  - Avoids over-reaction to mild overload conditions

39

Random early detection (RED) packet drop





40

Random early detection (RED) packet drop

- 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 …

41

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

42