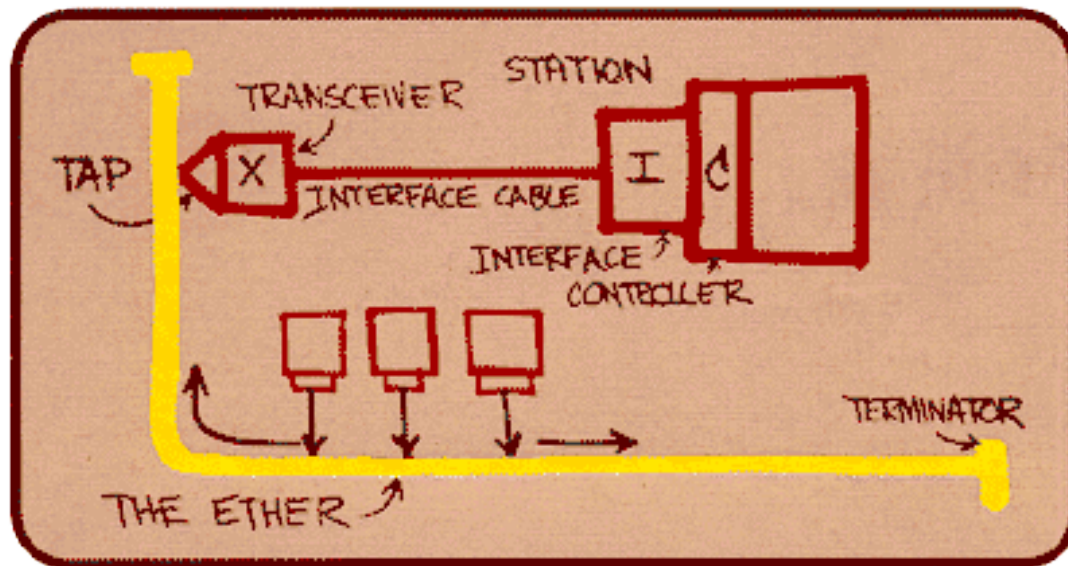


Randomization

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

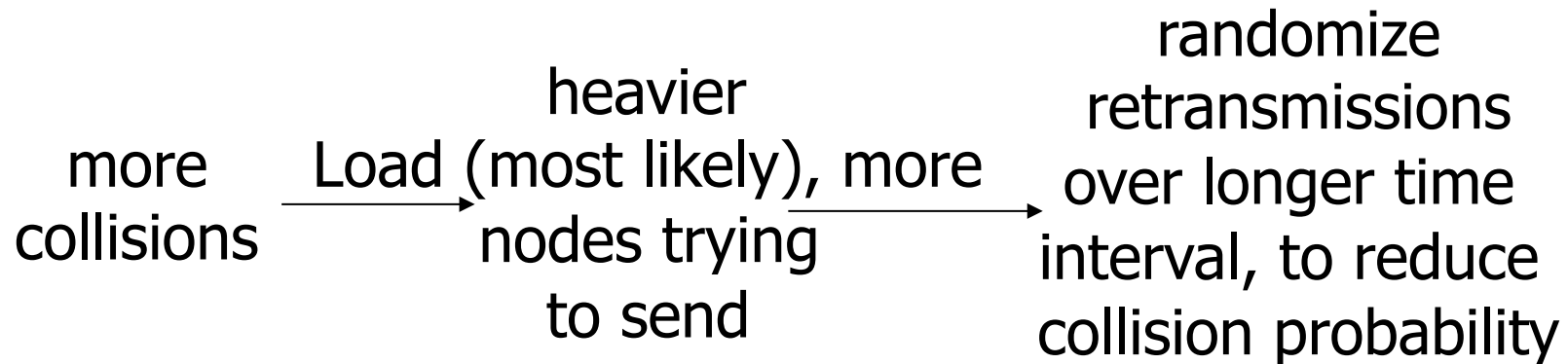
Jam Signal: Make sure all other transmitters are aware of collision; 48 bits;

Exponential Backoff:

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

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



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 backoff rather than something more sophisticated such as AIMD in TCP congestion control: simplicity
 - Note: Ethernet does multiplicative-increase-complete-decrease (why?)

The bottom line

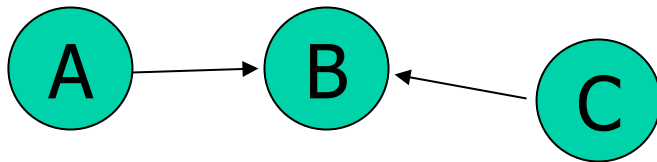
- ❑ Why does Ethernet use randomization: to desynchronize:

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

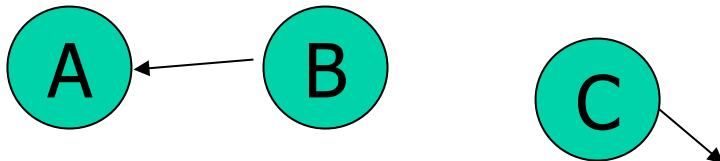
Excursion: What if wireless?!

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?

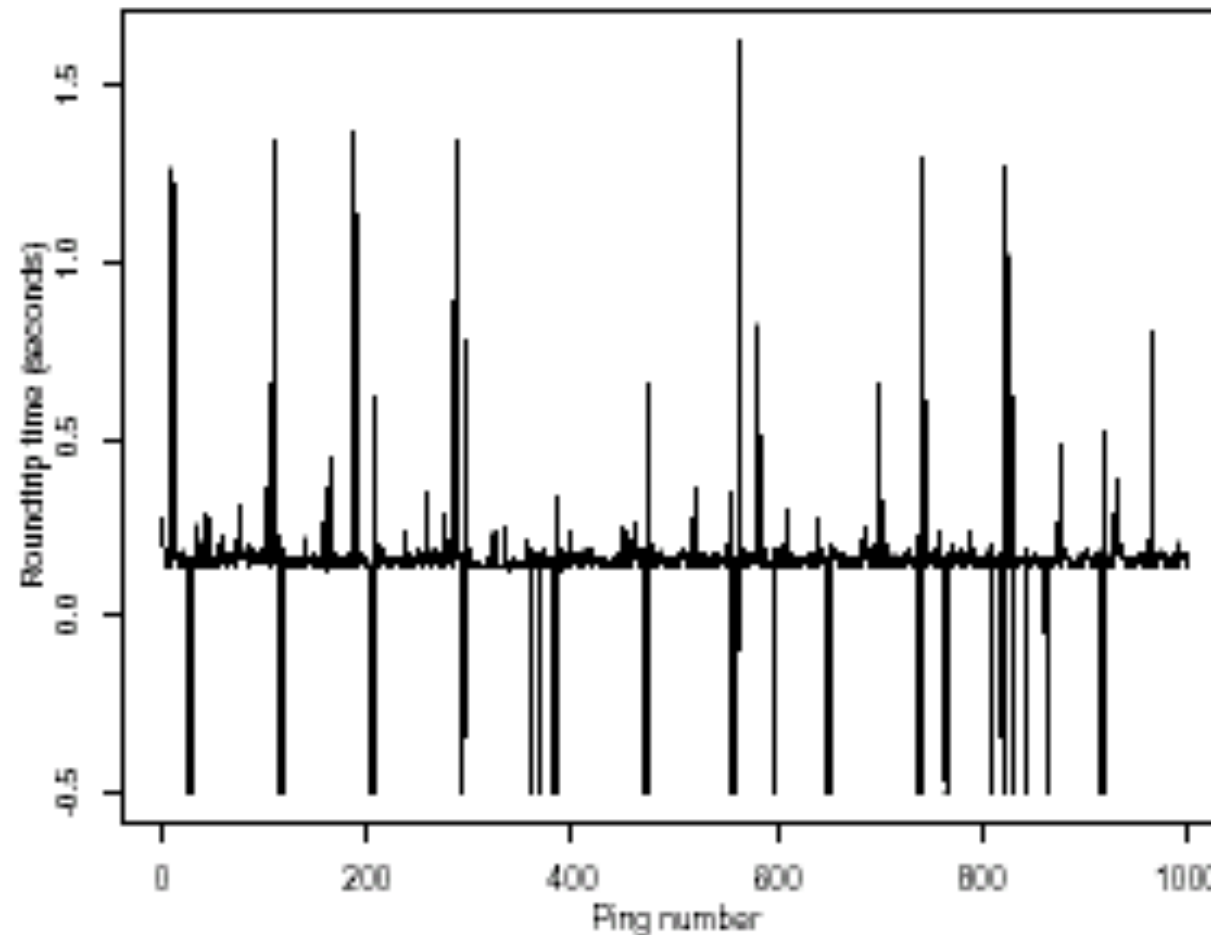
Therefore, CD is often replaced by (best effort) **Collision Avoidance** (CA, cf *DIFS/SIFS* etc.)

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 interference) time periods (e.g., the **Jade protocol**).

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

(de)Synchronization of periodic routing updates

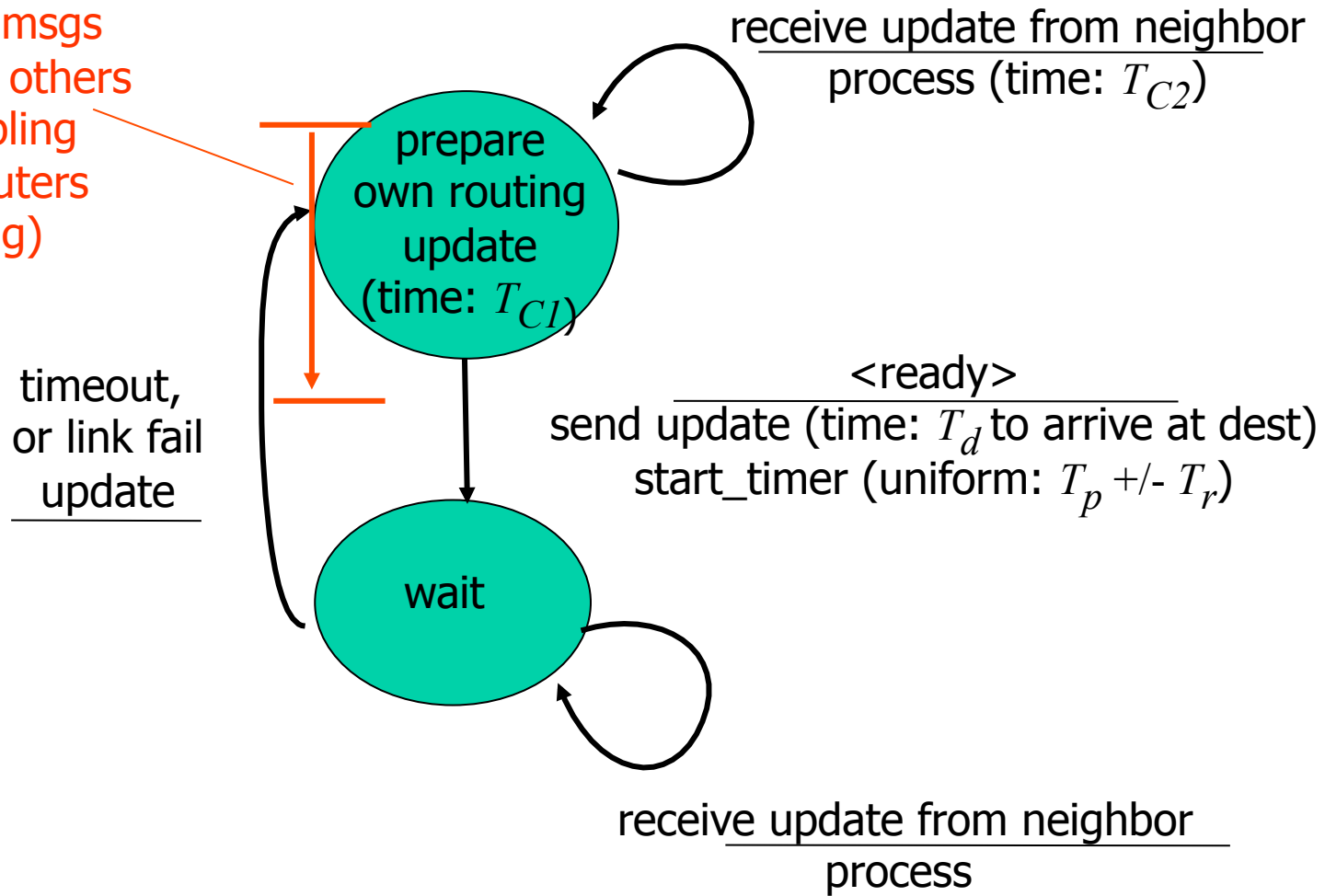
- ❑ Periodic losses observed in end-end Internet traffic
- ❑ Why?



source: Floyd,
Jacobson 1994

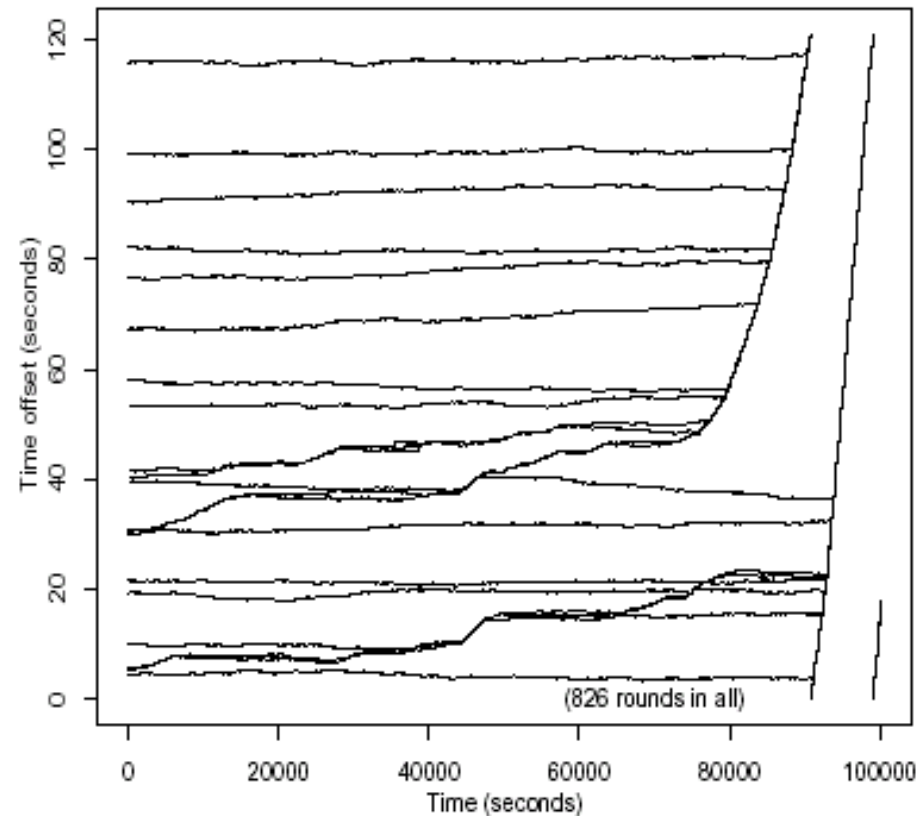
Router update operation:

time spent in state
depends on msgs
received from others
(weak coupling
between routers
processing)



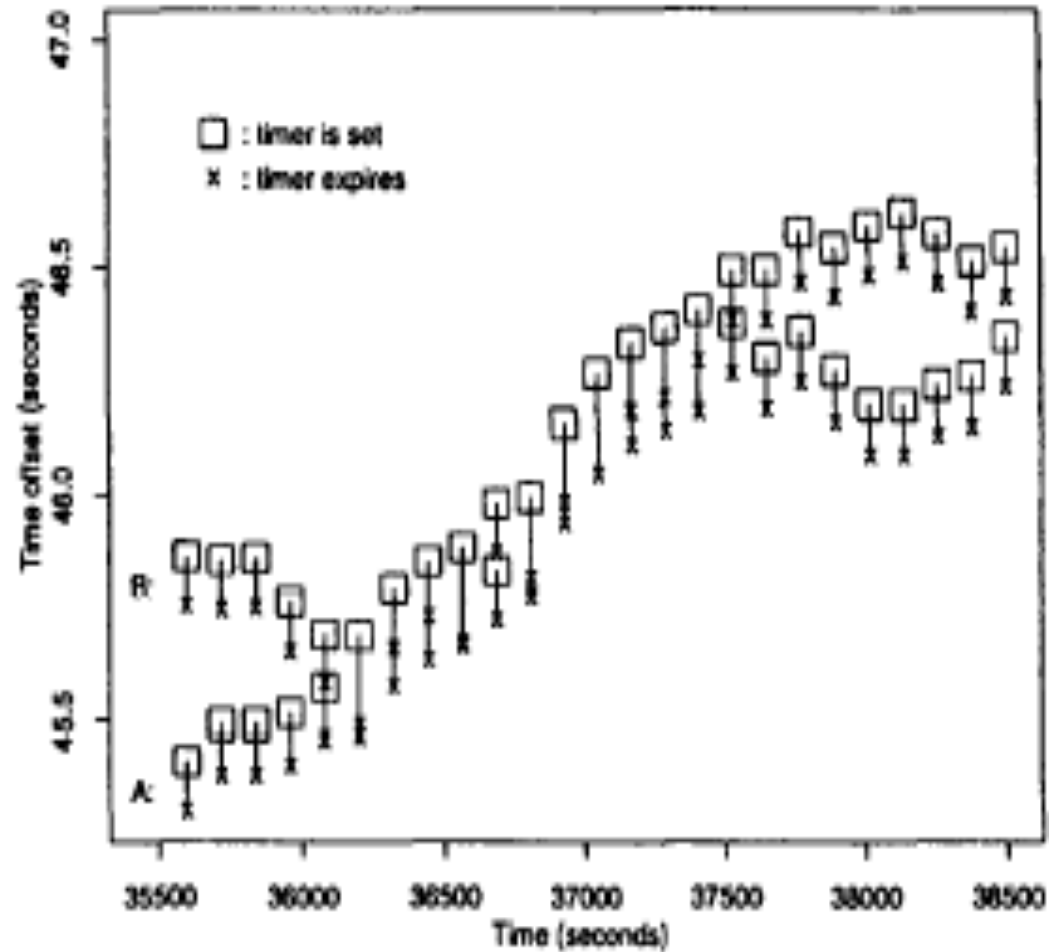
Router synchronization

- ❑ 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



Zooming in ...

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



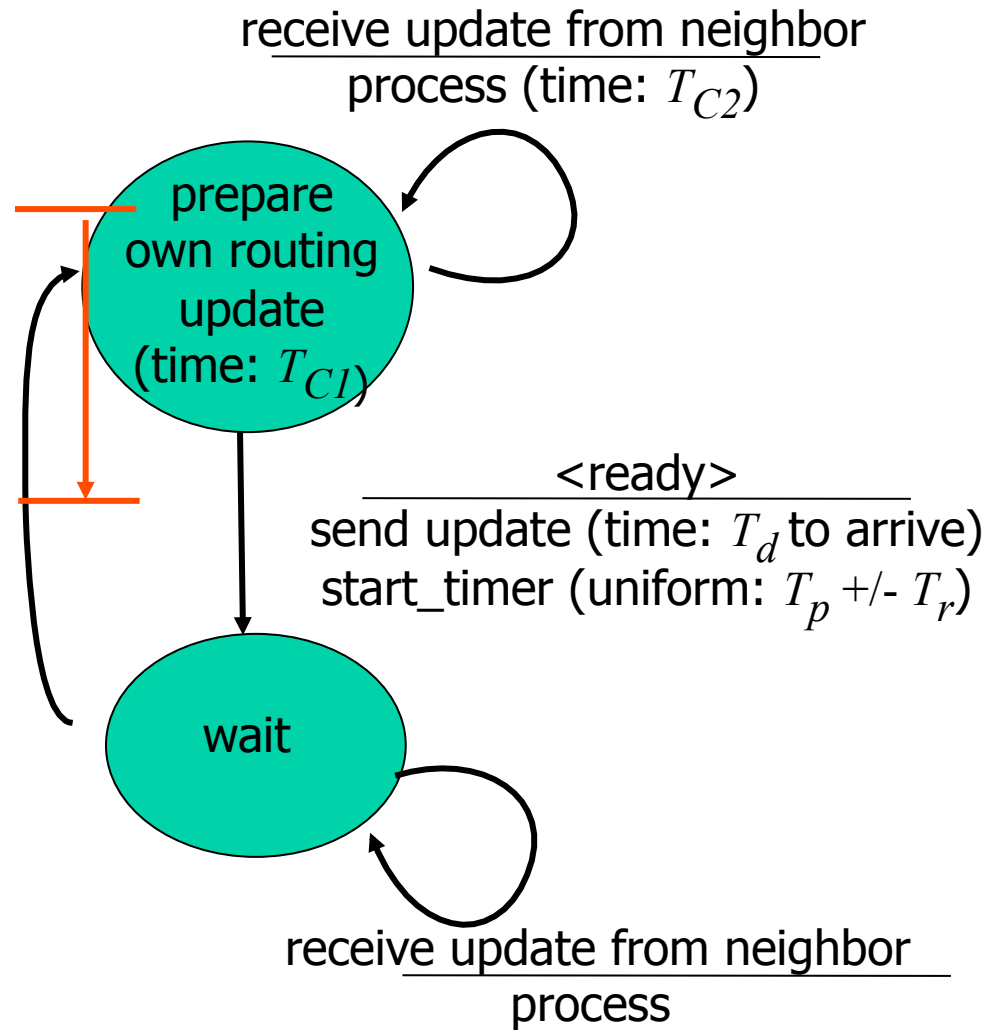
Sync

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

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

Avoiding synchronization

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

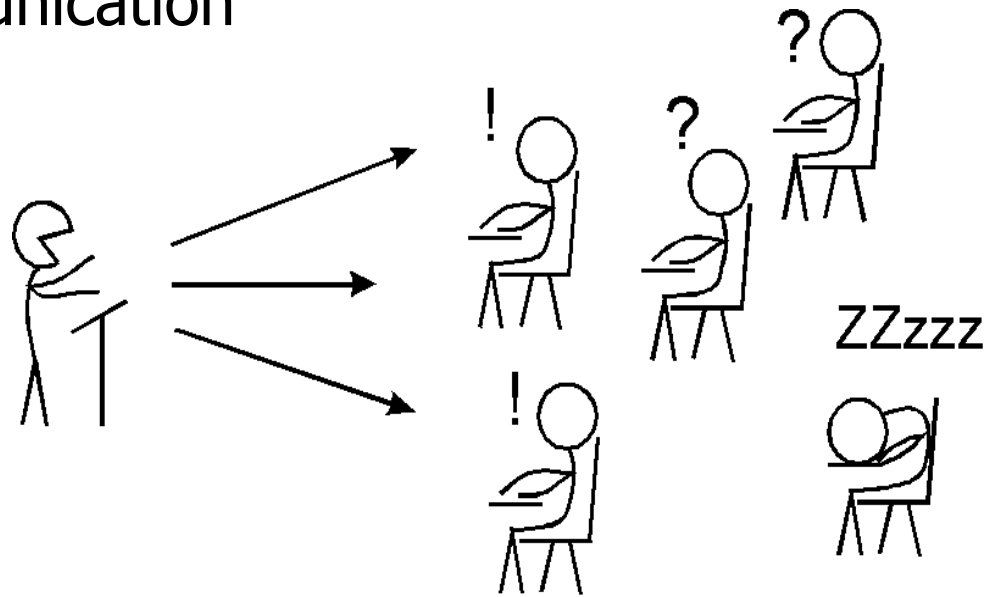


Router (de)synchronization

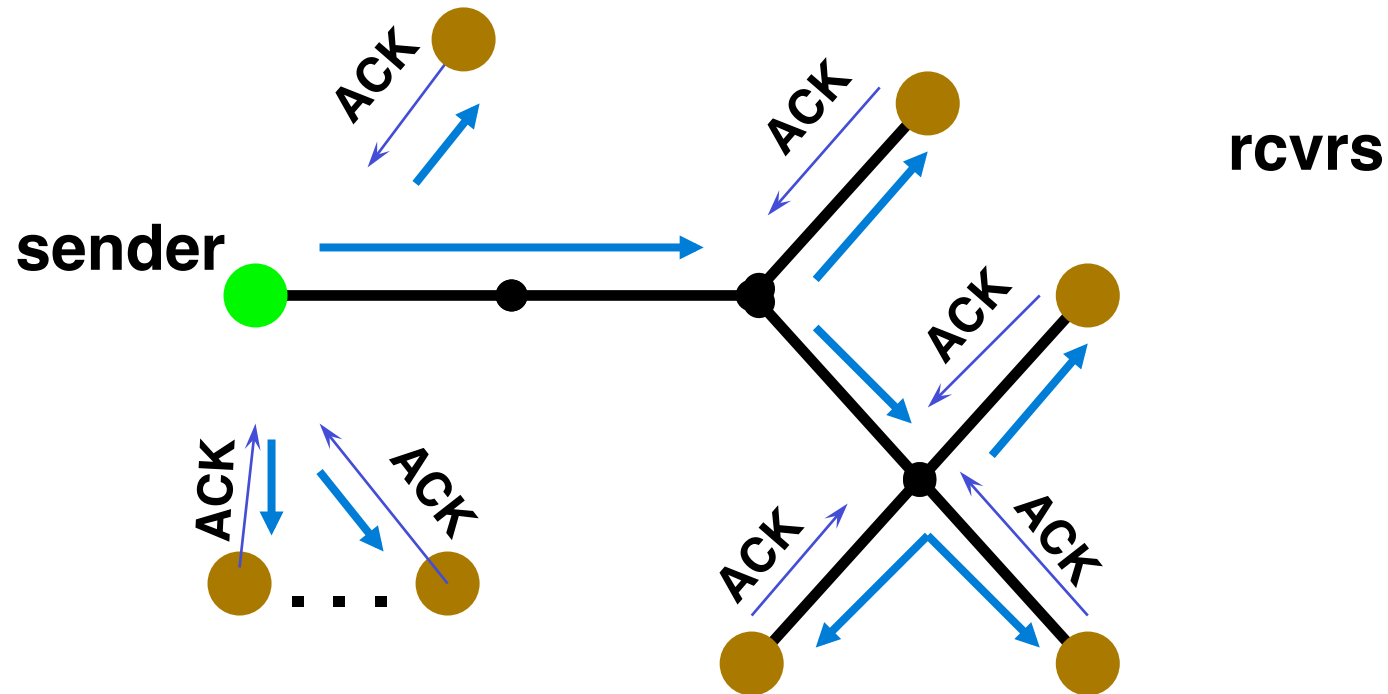
Use of randomization: Desynchronize routers!

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



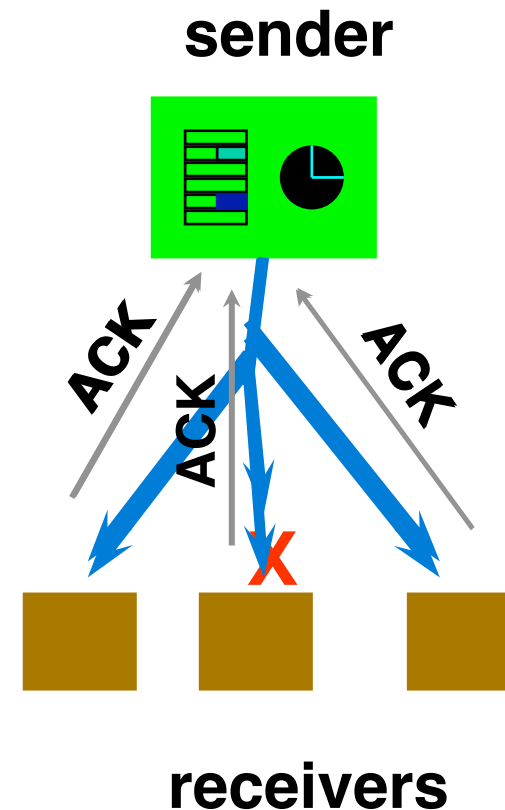
Sender Oriented Reliable Mcast

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

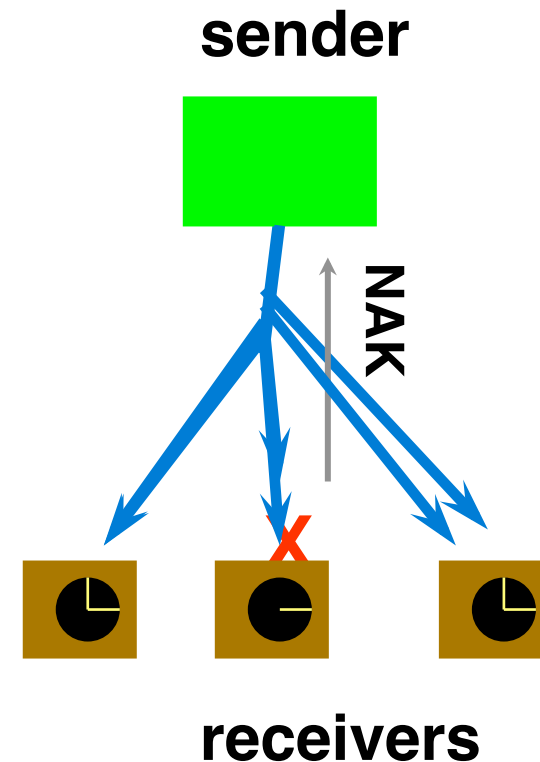
Sender:

- ❑ 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

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

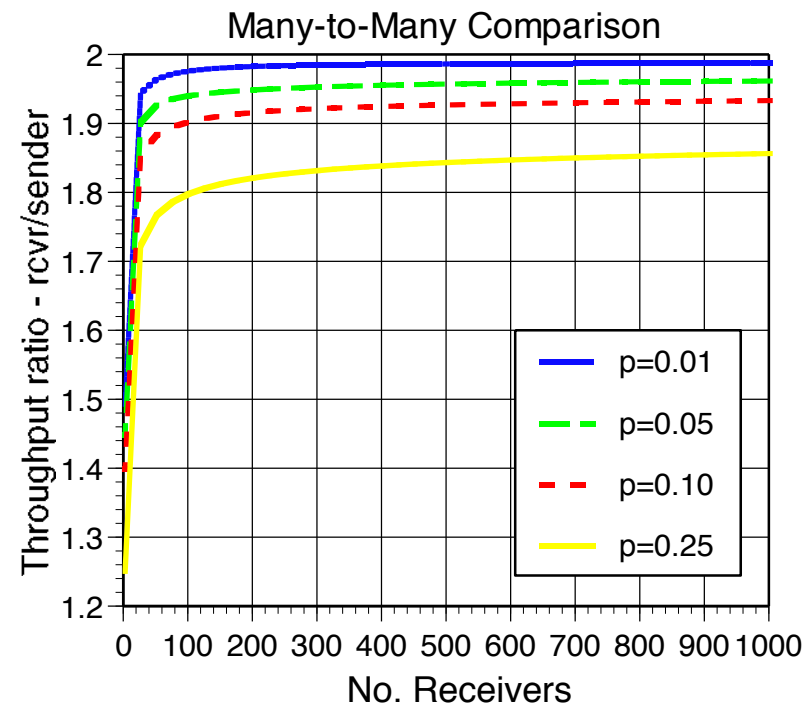
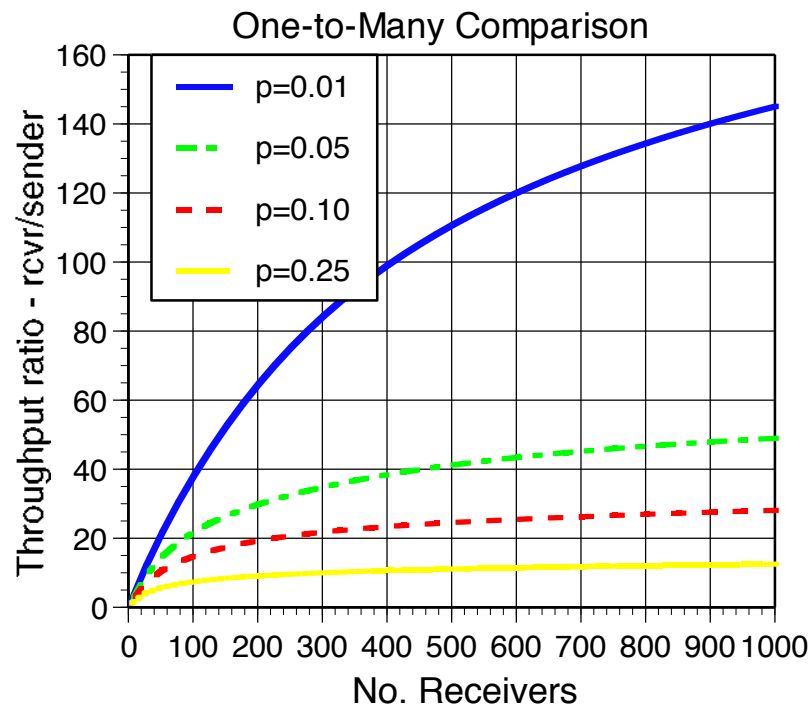
M - total number of transmissions per packet

$$P[M \leq m] = (1 - p^m)^R, \quad m = 1, \dots$$

$$E[M] = \sum_{m=0}^{\infty} 1 - (1 - p^m)^R$$

Sender vs. Receiver (cont.)

Metric - rcvr oriented thrupt/sender oriented thrupt



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

RM: Coping with Scale, Heterogeneity

Issues:

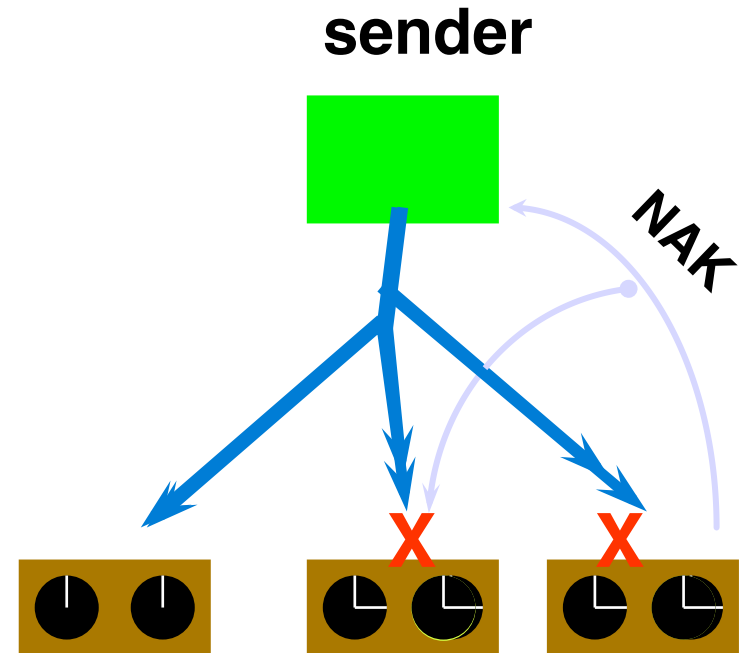
- ❑ Avoid feedback implosion in reverse path
- ❑ Avoid receiving unneeded data (retrans.) in forward path
- ❑ Recover data quickly, avoid long repair times

Techniques:

- feedback suppression
- local recovery

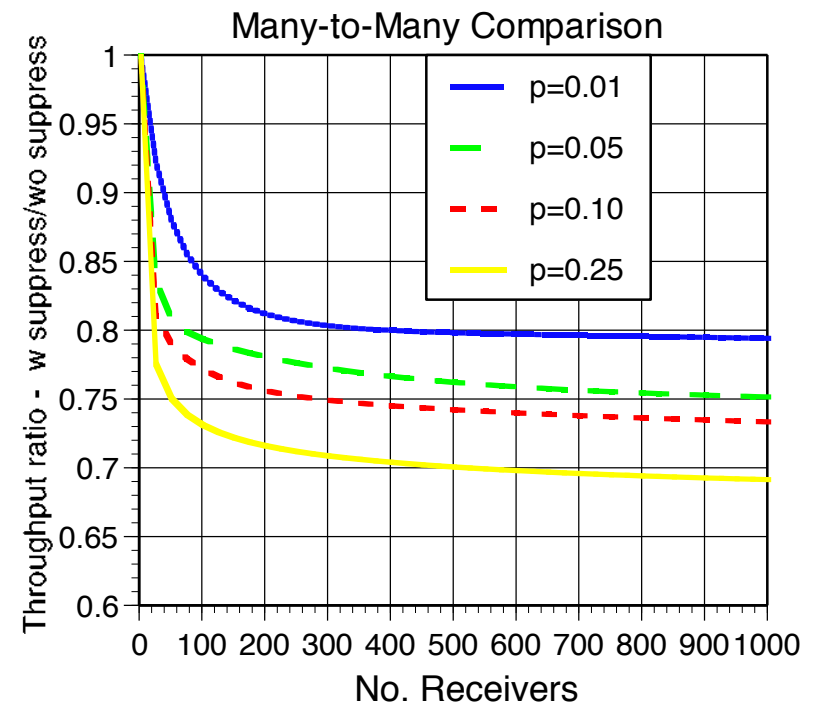
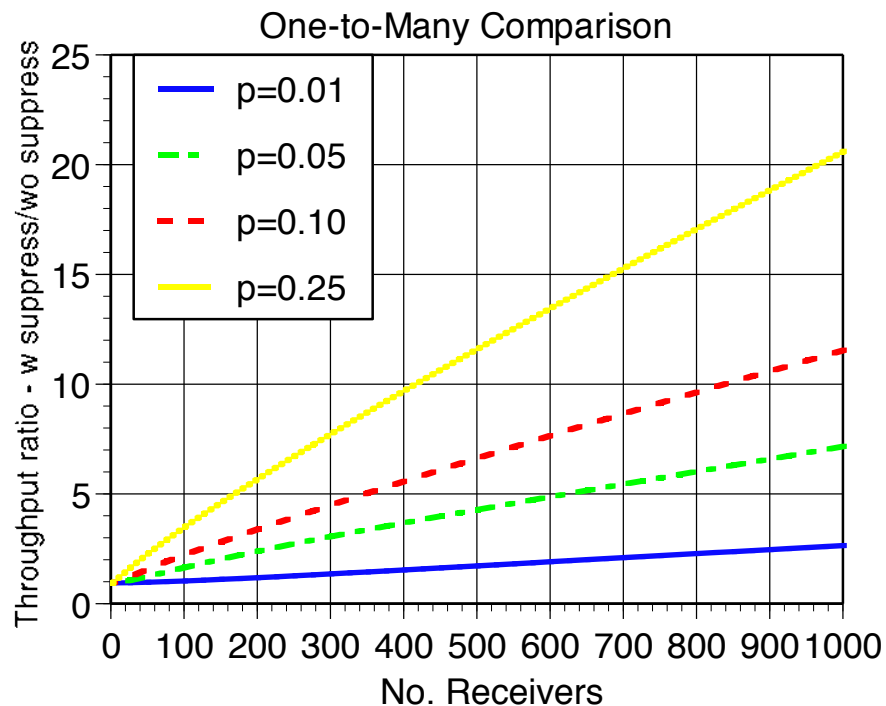
Feedback Suppression

- ❑ 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)



Feedback Suppression: Performance gains

Metric - suppression thruput/no suppression thruput



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

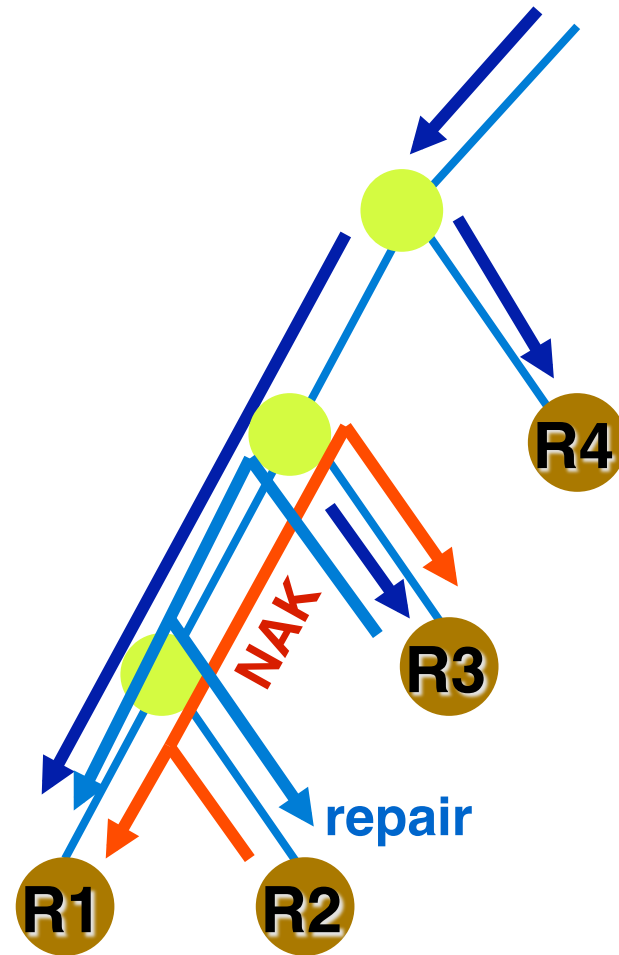
Local Recovery in SRM

- ❑ 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

“A Reliable Multicast Framework for Lightweight Sessions ...”, Floyd et al., ToN 1996

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



Reliable multicast (SRM)

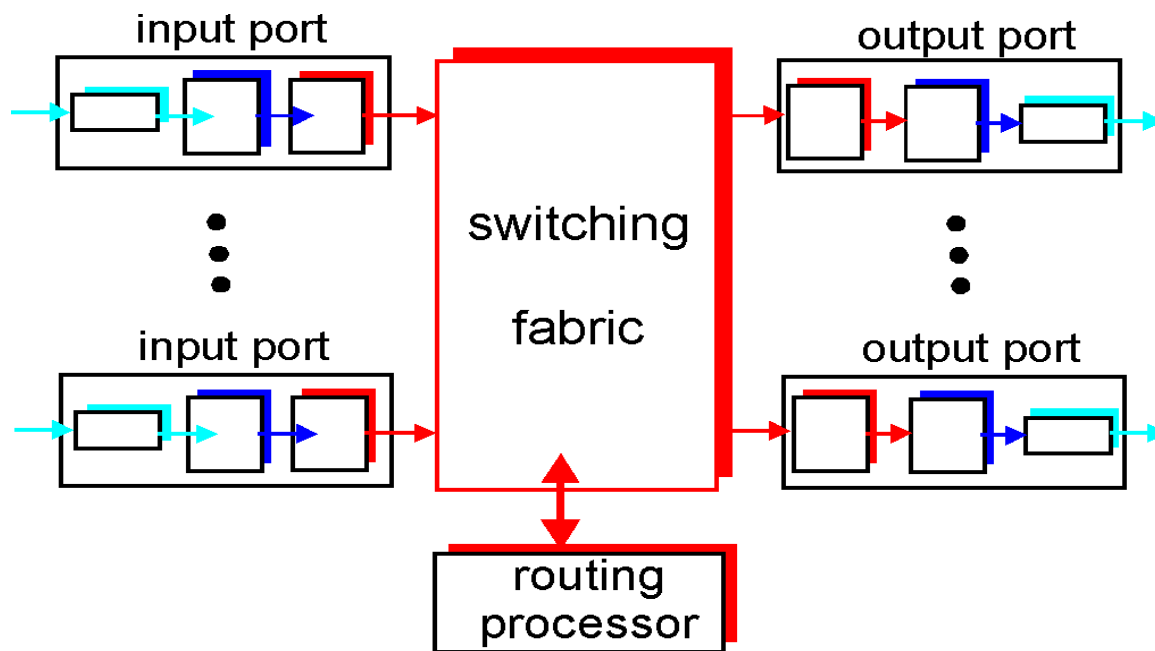
Use of randomization

- ❑ 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.

Randomization in Switch Scheduling

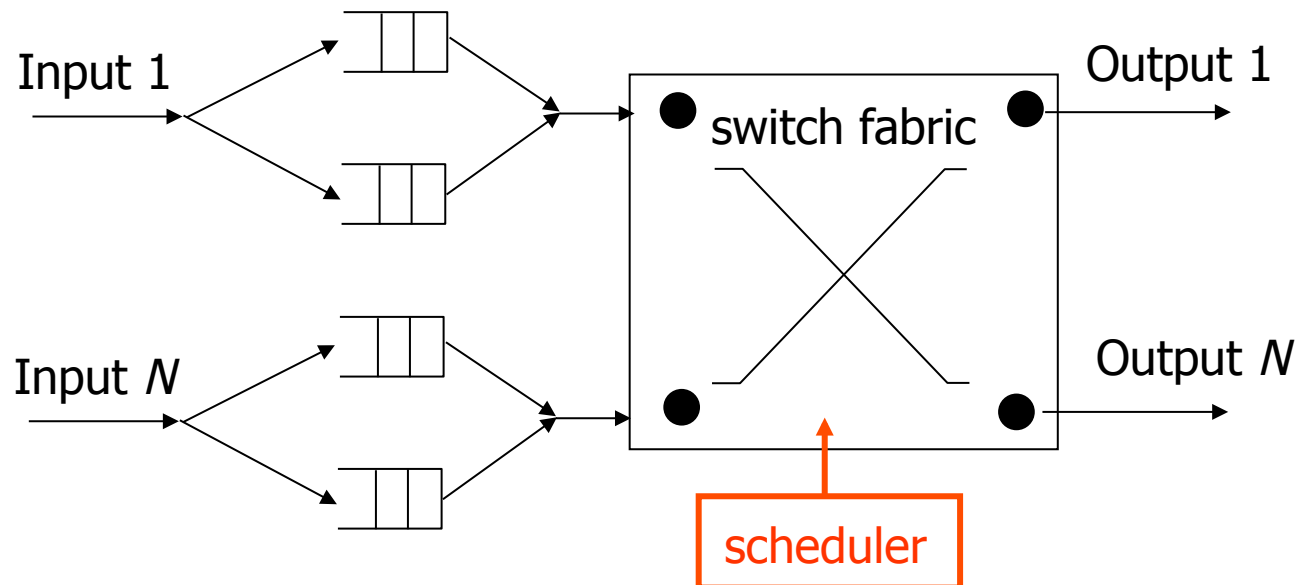
Two key router functions:

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



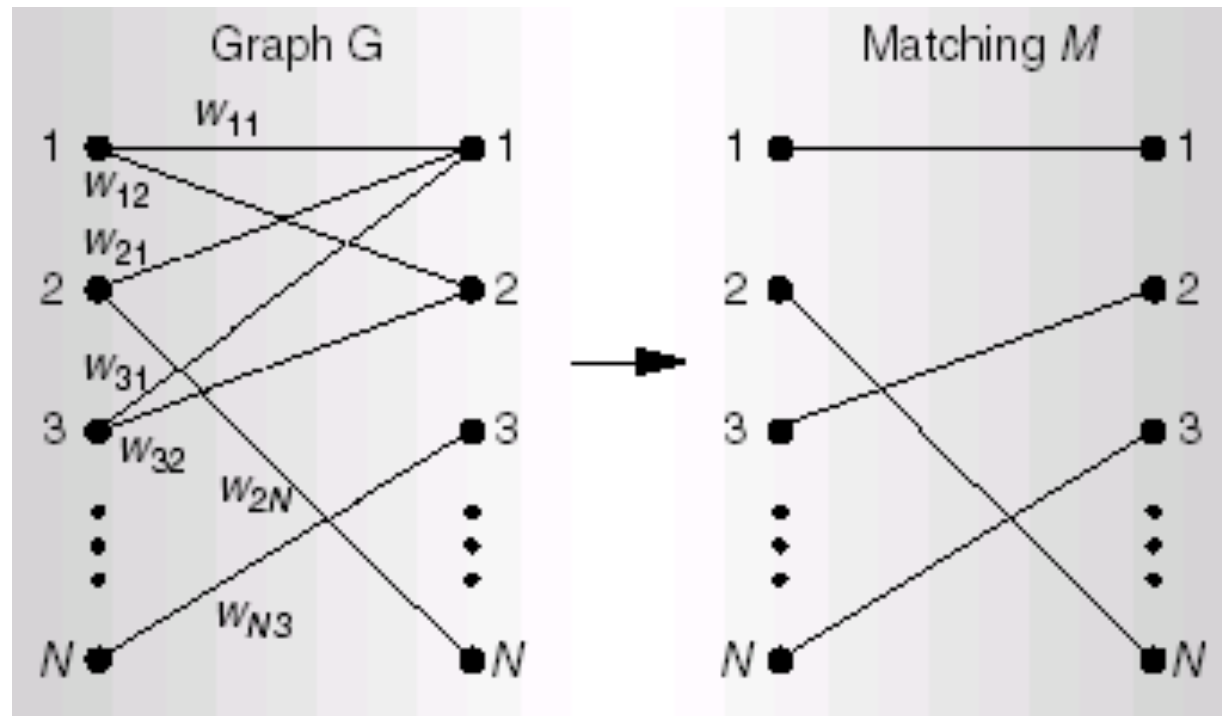
Input queueing with bufferless outputs

- 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



Solving the assignment problem

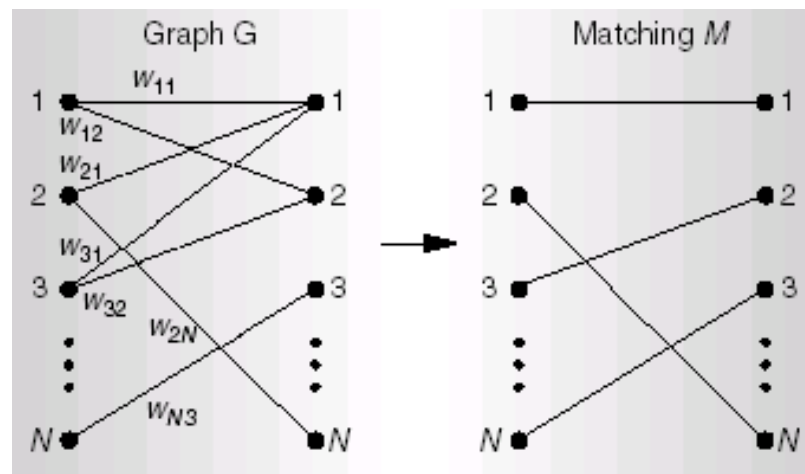
- ❑ W_{ij} : Queue length (weight) of input i , queue j
- ❑ **Maximum weight matching solution: $O(N^3)$** that can deliver 100% throughput
- ❑ Can we do better?



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

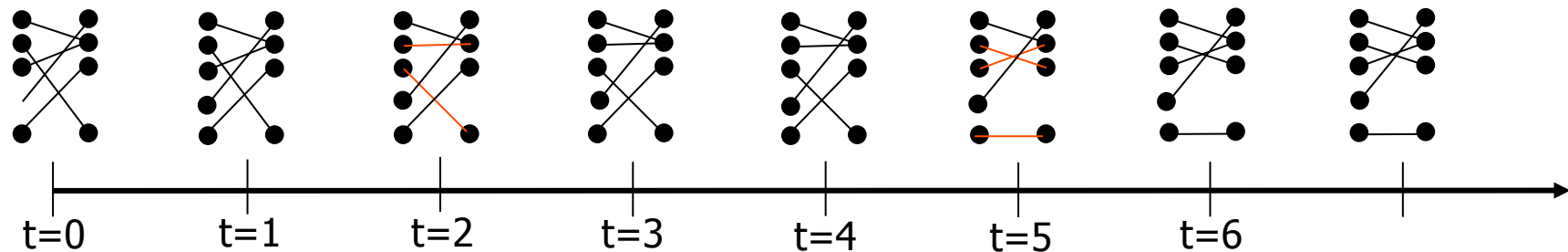


Using randomization to optimize

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

randomly
generated
solution
better (so
solution
changes)



Using randomization to optimize

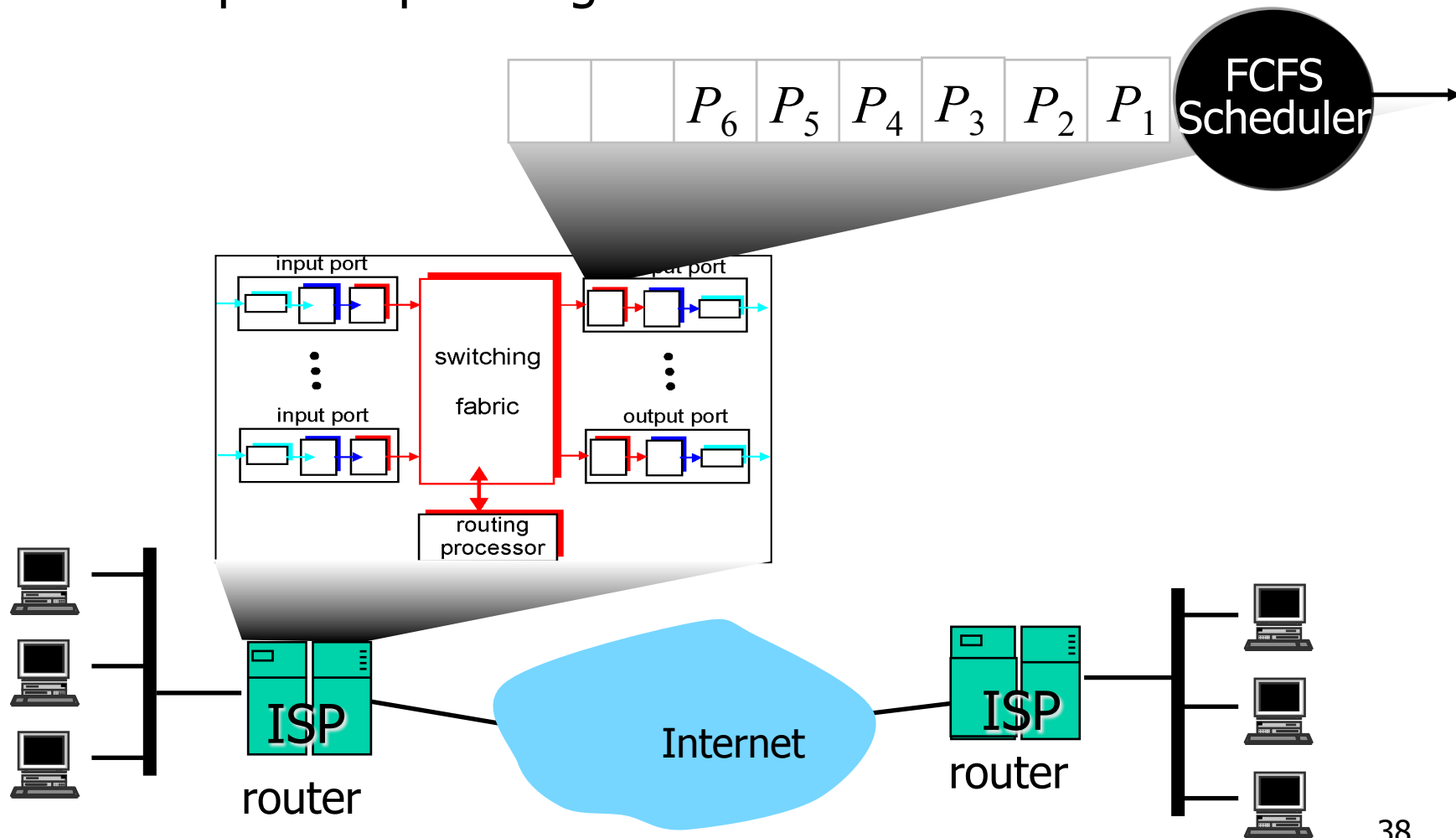
Result [Tassiulas;Shah et al.]: Randomized algorithm achieves 100% throughput (i.e., as good as brute force, non-randomized $O(N^3)$ optimization algorithm).

“Linear complexity algorithms for maximum throughput in ... input queued switches”,
Tassiulas, 1998

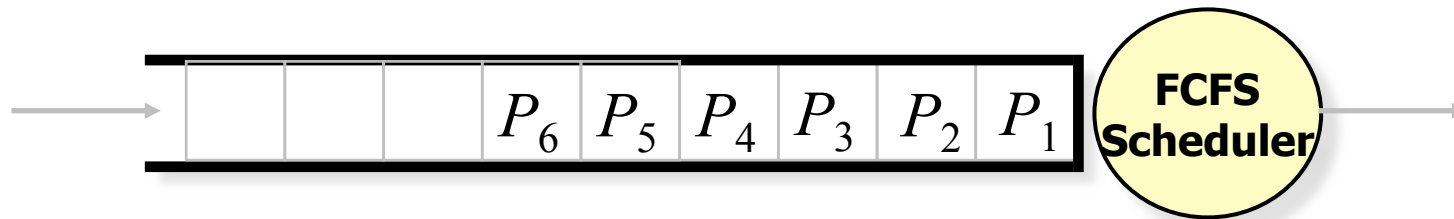
“An Efficient Randomized Algorithm for Input-Queue Switch Scheduling”, Shah et al., 2001

Randomization in Router Queue Management

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

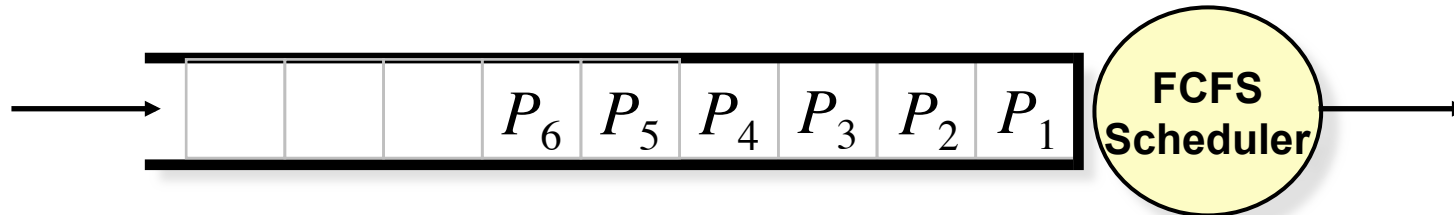


The case against drop-tail queue management



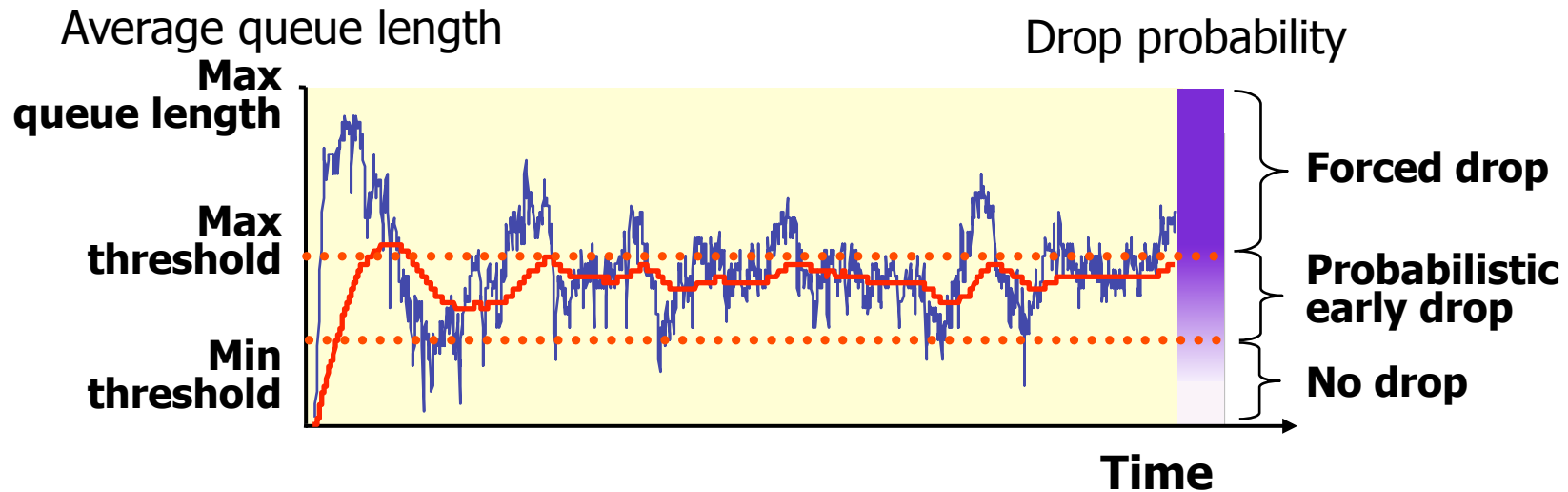
- ❑ Large queues in routers are “a bad thing”
 - End-to-end latency dominated by length of queues at switches in network
- ❑ Allowing queues to overflow is “a bad thing”
 - Connections transmitting at high rates can starve connections transmitting at low rates
 - Connections can *synchronize* their responses to congestion

Idea: Early random packet drop



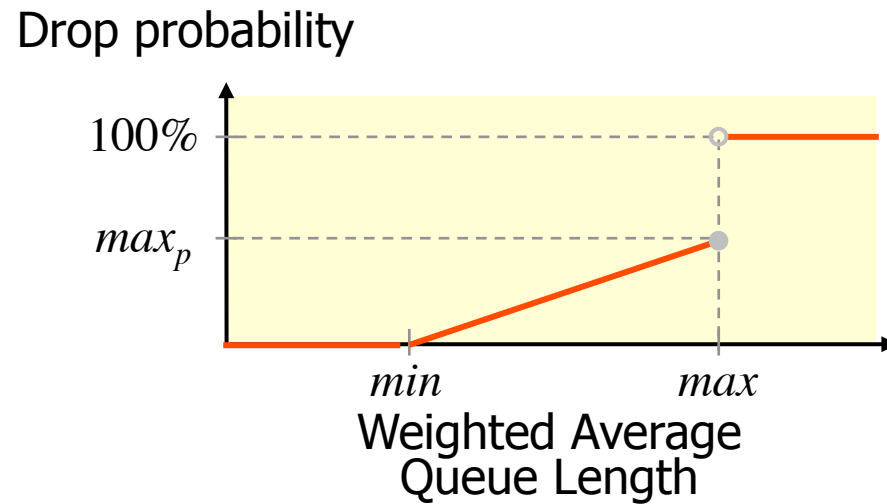
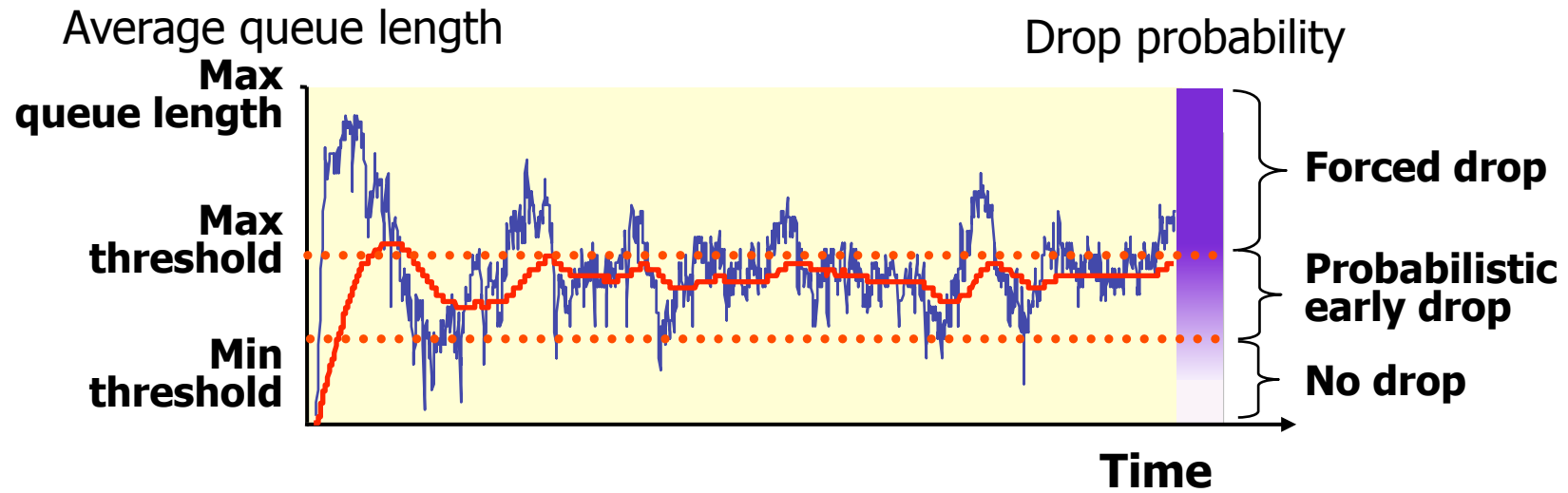
- When queue length exceeds threshold, packets dropped with fixed *probability*
 - Probabilistic packet drop: Flows see same loss *rate*
 - Problem: Bursty traffic (burst arrives when queue is nearly full) can be over-penalized

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

Random early detection (RED) packet drop



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 ...

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

- ❑ Randomization used in many protocols:
 - CSMA/CD: to desynchronize senders
 - BGP: to desynchronize routing updates
 - SRM: feedback suppression
 - Switch scheduling: performance increase
 - Active queue management