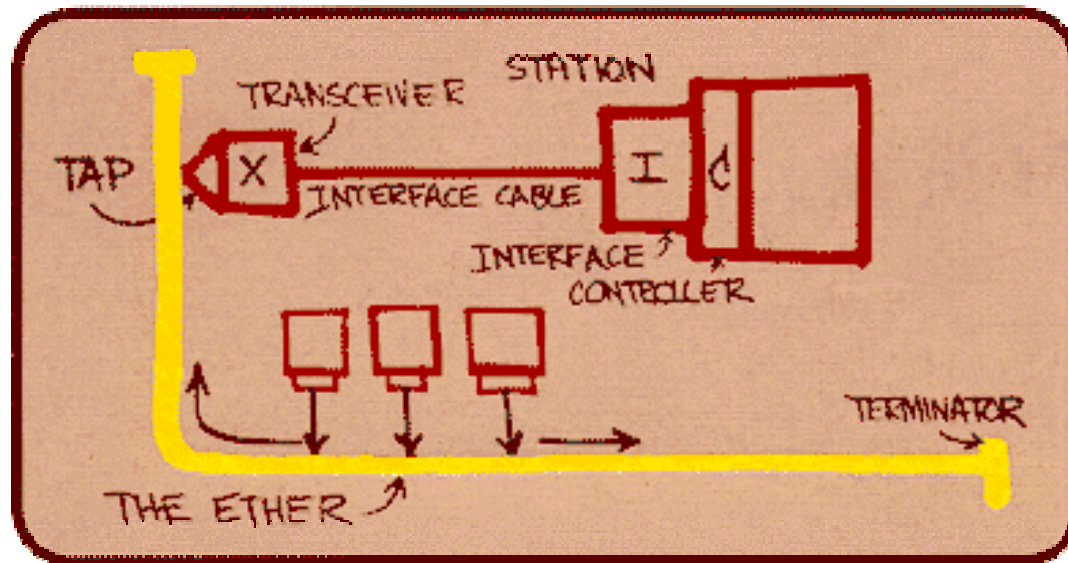


# 4. Randomization

- randomization used in many protocols
- we'll study examples:
  - Ethernet multiple access protocol
  - avoiding (unwanted) synchronization
  - active queue management
  - BitTorrent load balancing
  - power of two (random) choices: load balancing
  - randomized two-hop routing

# 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

# Ethernet: uses CSMA/CD

A: sense channel, if idle

then {

transmit, monitor channel;

If detect another transmission

then {

abort and send jam signal;

update # collisions;

delay according to exponential backoff algorithm;

goto A

}

else {done with frame; set collisions to zero}

}

else {wait until ongoing transmission over, goto A}

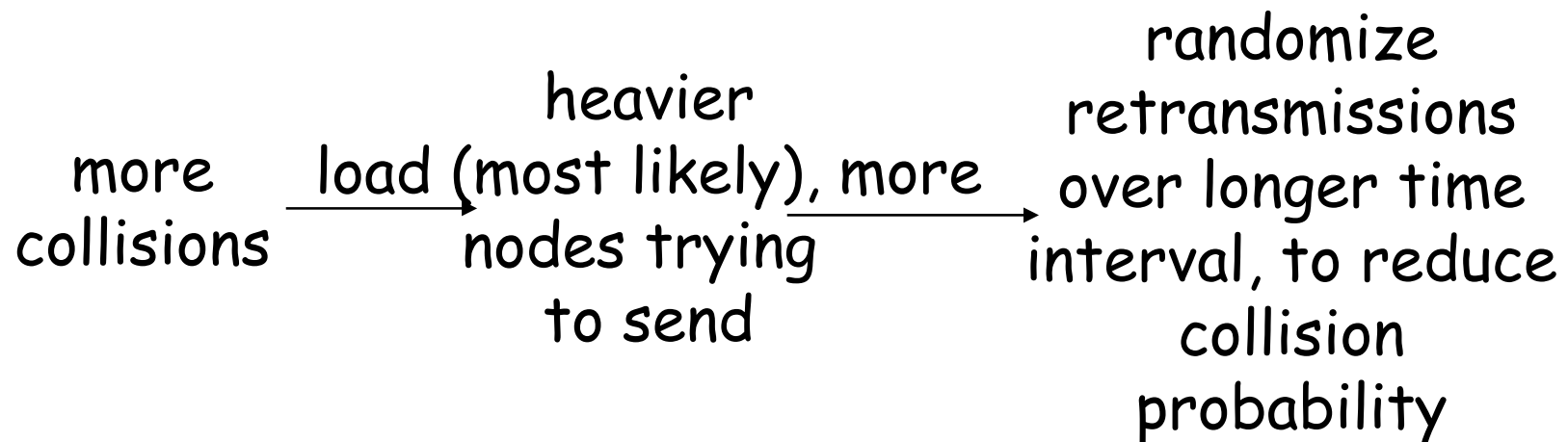
# Ethernet's CSMA/CD (more)

## 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  $K$  (and keep doubling on collisions until.....)
- ❑ 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 TCP's AIMD: simplicity
  - note: Ethernet does multiplicative-increase-complete-decrease

# Analyzing the CSMA/CD Protocol

**Goal:** quantitative understanding of performance of CSMA protocol

- ❑ fixed length pkts
- ❑ pkt transmission time is unit of time
- ❑ *throughput S* - number of pkts successfully (without collision) transmitted per unit time
- ❑  $\alpha$  - end-to-end propagation time
  - time during which collisions can occur

# Traffic model for Ethernet analysis

□ *offered load*  $G$  - number pkt transmissions attempted per unit time

○ *note*:  $S < G$ , but  $S$  depends on  $G$

○ *Poisson model*: probability of  $k$  pkt transmission attempts in  $t$  time units

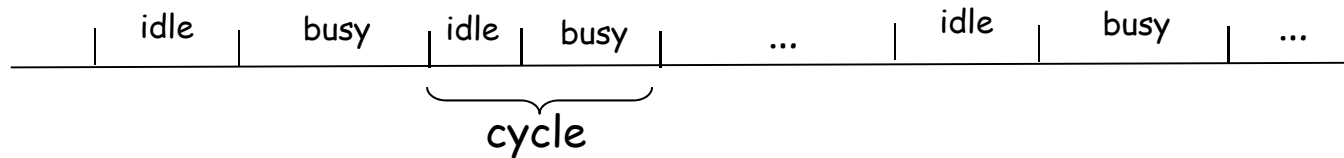
$$\text{Prob}[k \text{ trans in } t] = ((Gt)^k)(e^{-Gt})/k!$$

○ infinite population model

□ *capacity of multiple access protocol*:  
maximum value of  $S$  over all values of  $G$

# Analyzing CSMA/CD

Channel activity: cycles of idle, busy periods:



I - length of idle period, B - length of busy period;

$C = I + B$ : length of cycle

Successive cycle lengths independent, due to Poisson assumption

$$S = p/E[C] = p/(E[I]+E[B])$$

p - prob of successful transmission during busy period

$$p = p(0 \text{ transmissions in } \alpha) = e^{-\alpha G}$$

- each busy period begins with transmission
- transmission beginning busy period successful if no other transmissions in  $\alpha$

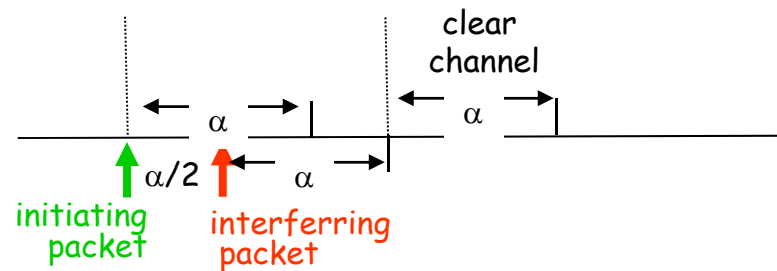
# Analyzing CSMA/CD(cont)

I exponentially distributed, mean  $1/G$  ( $E[I] = 1/G$ )

Computing  $E[B]$ :

Case 1: no collision (NC):  $E[B|NC] = 1 + \alpha$  and  $P(NC) = e^{-\alpha G}$

Case 2: collision (C):  $P(C) = 1 - e^{-\alpha G}$



Hand-wavy argument to compute  $E[B]$ :

Consider most likely case: one other transmission interferes with pkt starting busy period

Interfering packet arrives on average  $\alpha/2$  after start of busy period

Interfering packet transmits for a time units until collision detected (worst case)

Additional  $\alpha$  time needed for channel to become idle

Therefore  $E[B|C] = 5\alpha/2$

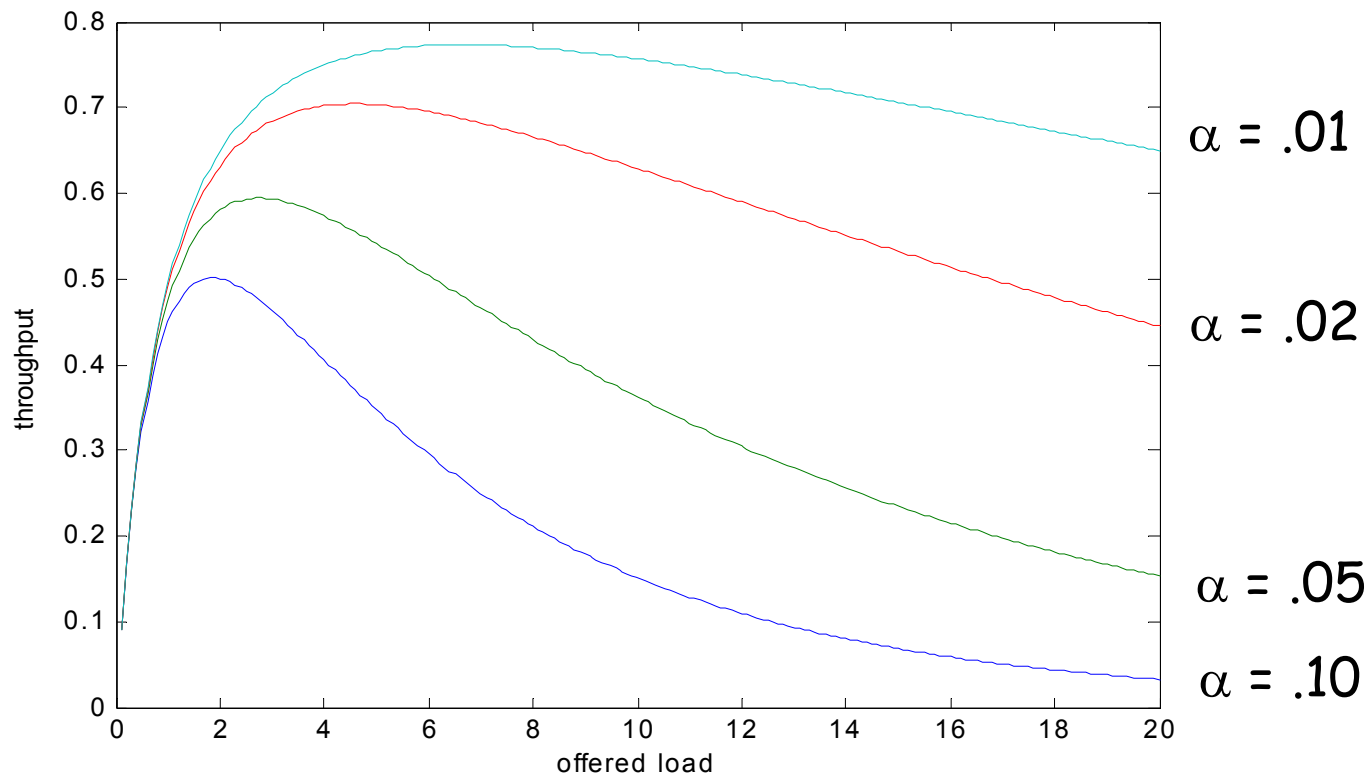
# Analyzing CSMA/CD(cont)

$$\begin{aligned} E[C] &= P(\text{NC}) E[B|\text{NC}] + P(\text{C}) E[B|\text{C}] \\ &= e^{-\alpha G} (1 + \alpha) + (1 - e^{-\alpha G}) 5\alpha/2 \end{aligned}$$

and

$$\begin{aligned} S &= p / (E[I] + E[B]) \\ &= e^{-\alpha G} / (1/G + e^{-\alpha G} (1 - 3\alpha/2) + 5\alpha/2) \end{aligned}$$

# Ethernet: impact of load, $\alpha$ on performance

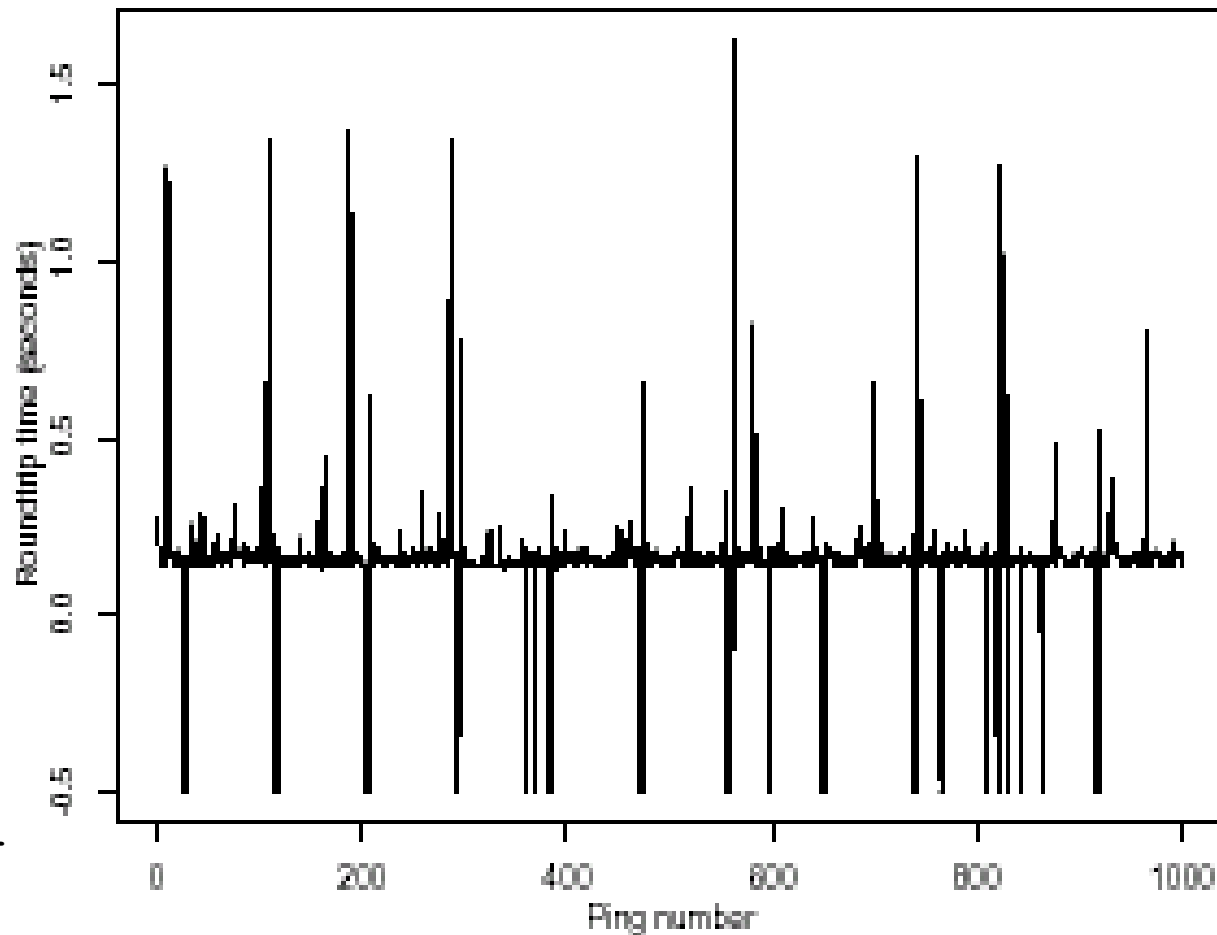


# The bottom line

- 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

# (de)Synchronization of periodic routing updates

- periodic losses observed in end-end Internet traffic

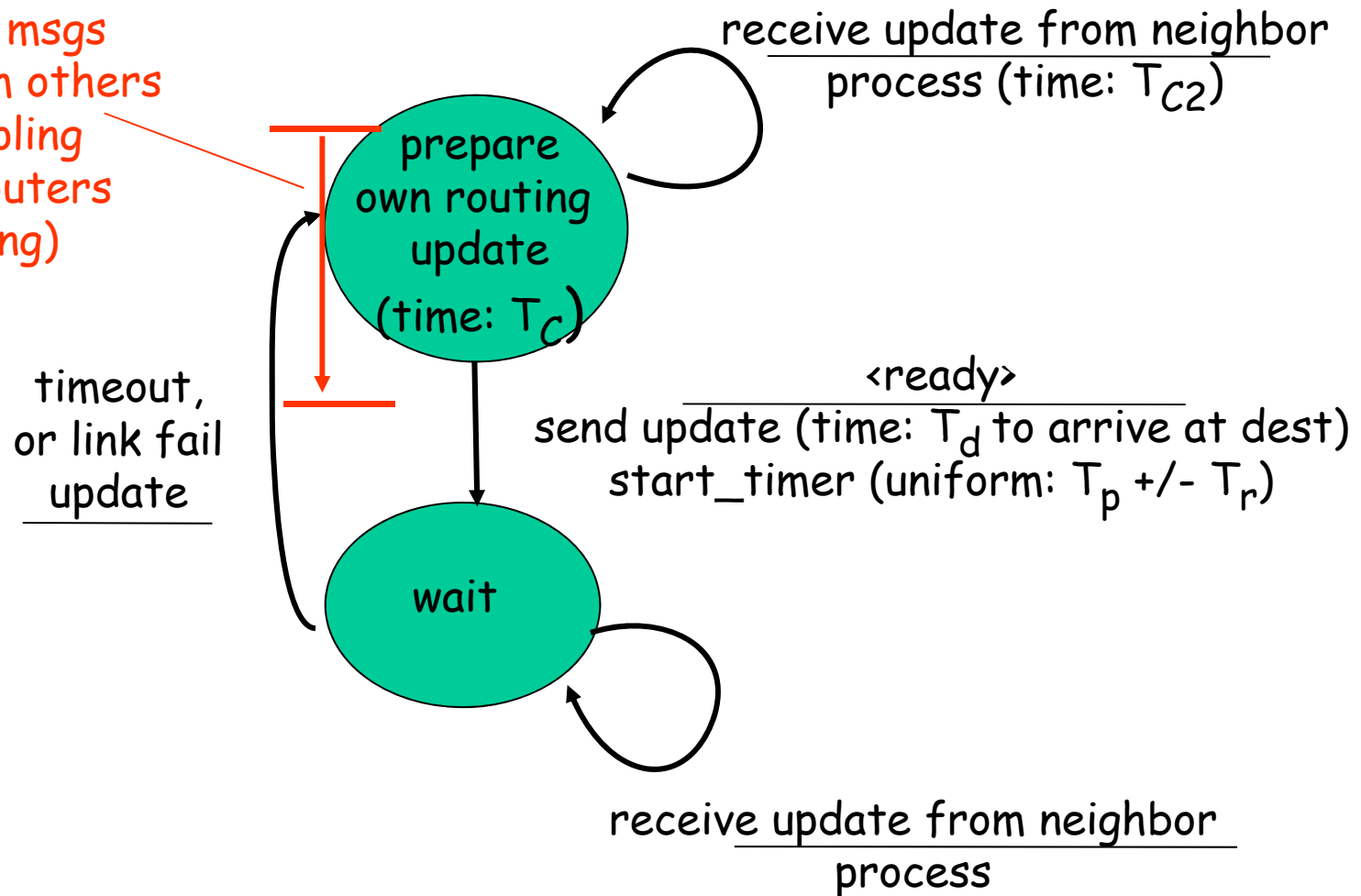


source: Floyd,  
Jacobson 1994

Why?

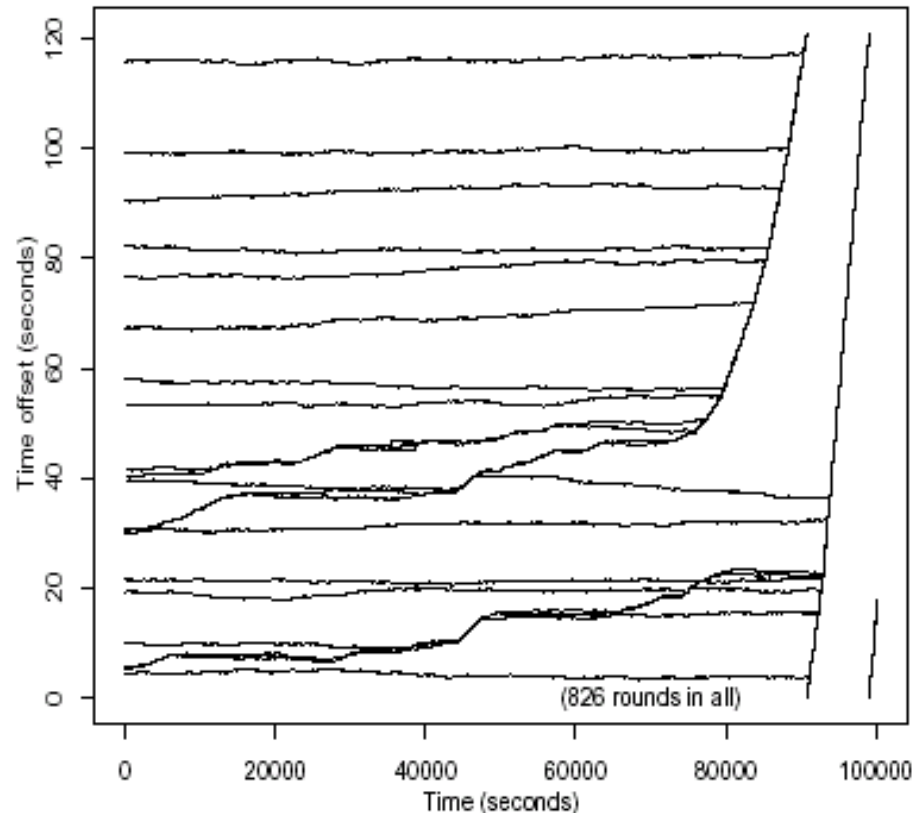
# Router update operation

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

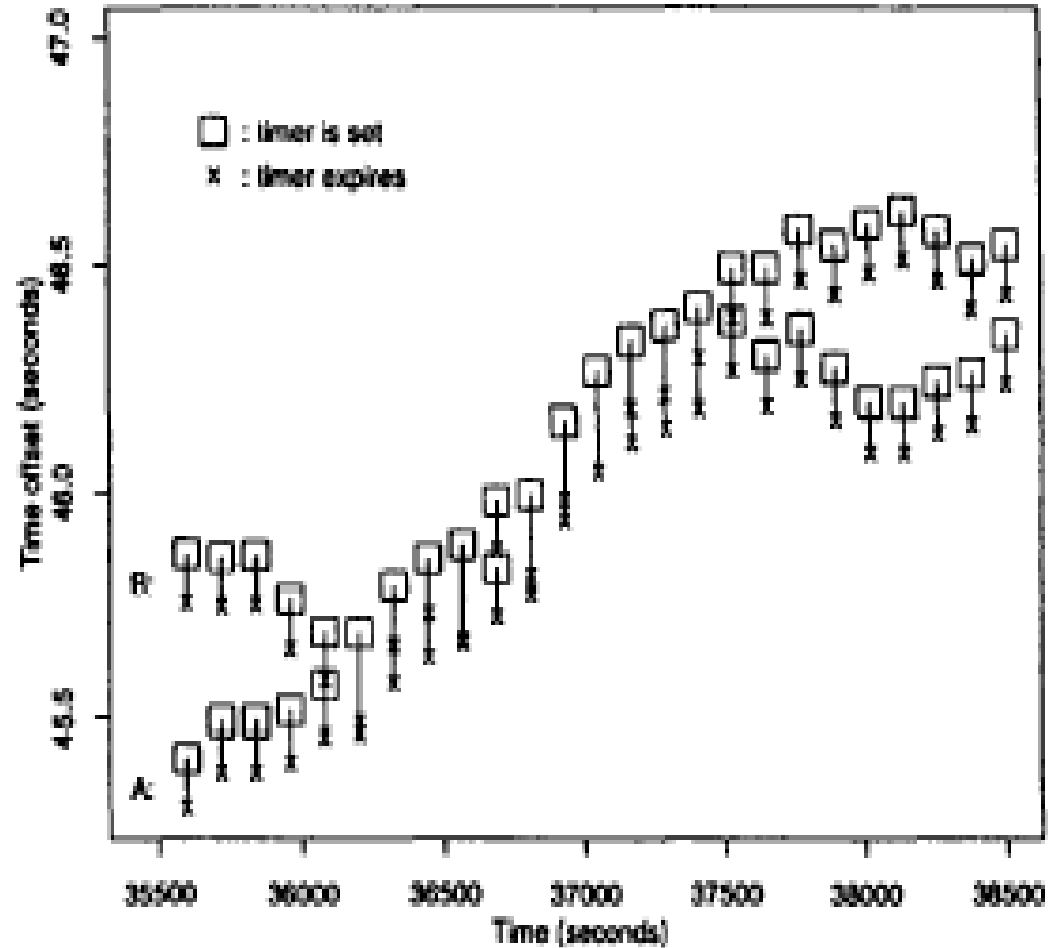


# Router synchronization

- ❑ 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

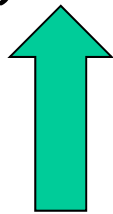


- blowup of previous graph
- note expansion of computation phase  
→ increased period

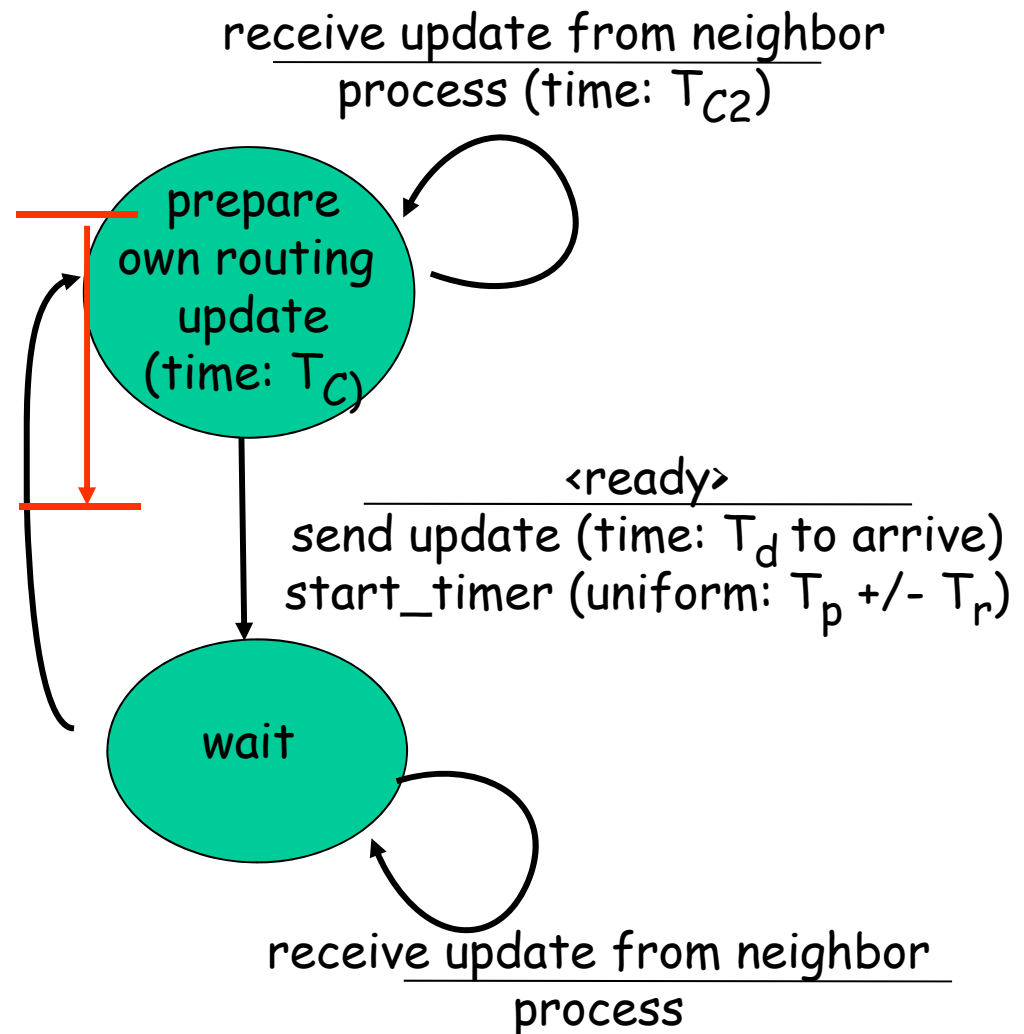


# Avoiding synchronization

- choose random timer component,  $T_r$  large (e.g., several multiples of  $T_c$ )



Add enough randomization to avoid synchronization



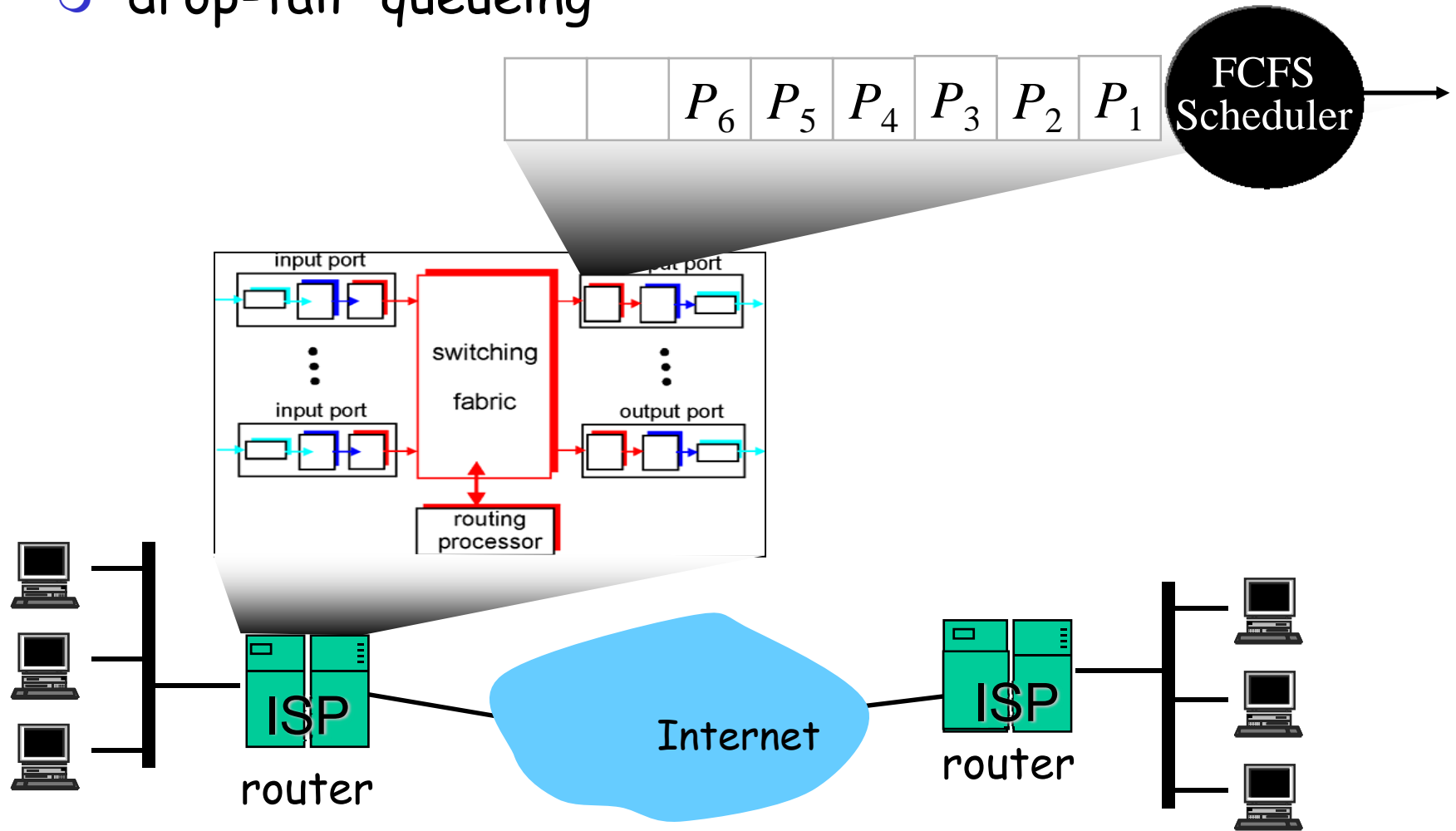
# Sync

- ❑ coupled routers
- ❑ example of spontaneous synchronization
  - fireflies
  - sleep cycle
  - heart beat
  - more?

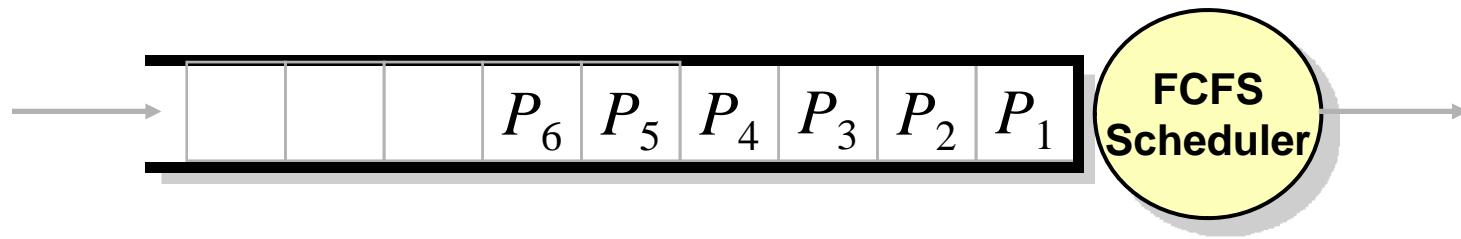
Steven Strogatz . *Sync*, Hyperion Books, 2003.

# Randomization in Router Queue Management

- normally, packets dropped only when queue overflows
  - “drop-tail” queueing

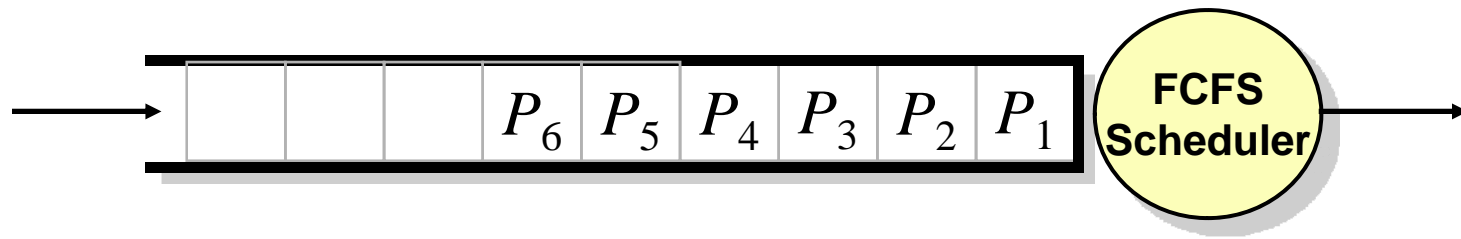


# The case against drop-tail queue management



- ❑ 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 response to congestion

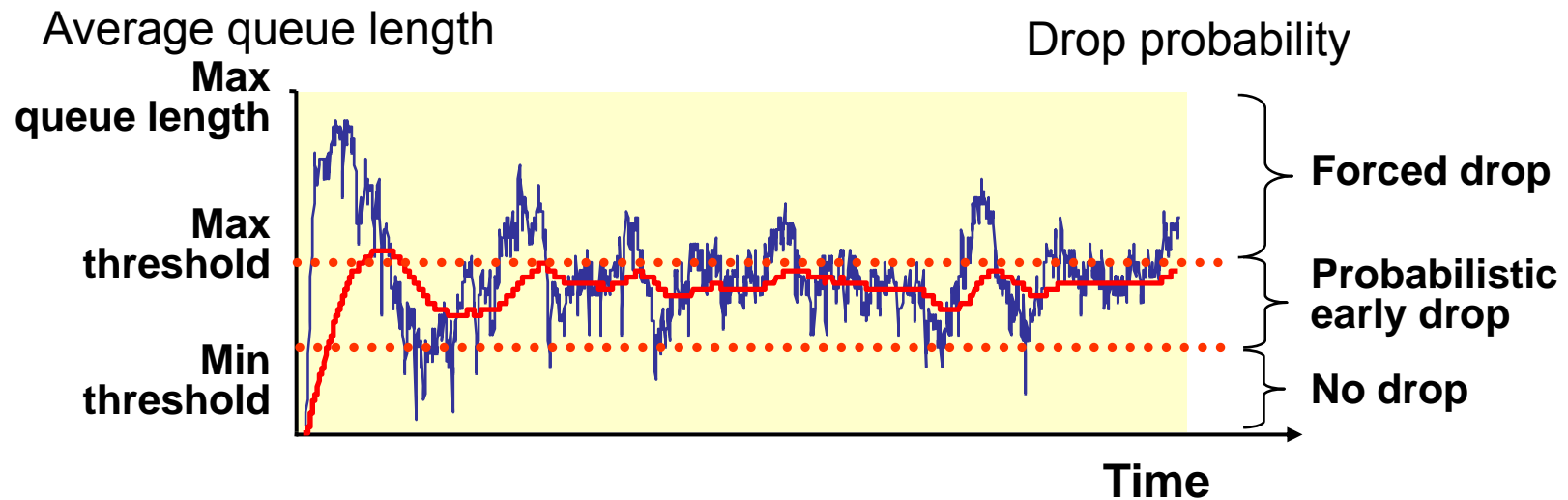
# Idea: early random packet drop



When queue length exceeds threshold, drop packets with queue length dependent *probability*

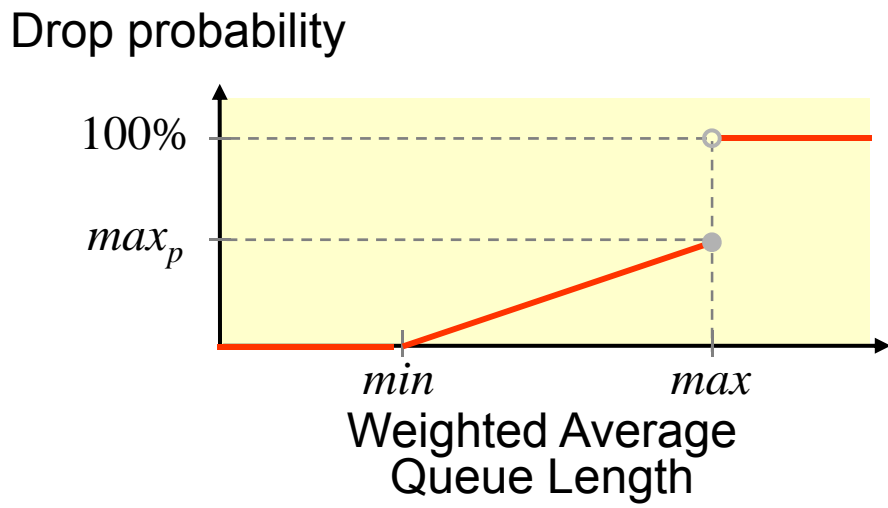
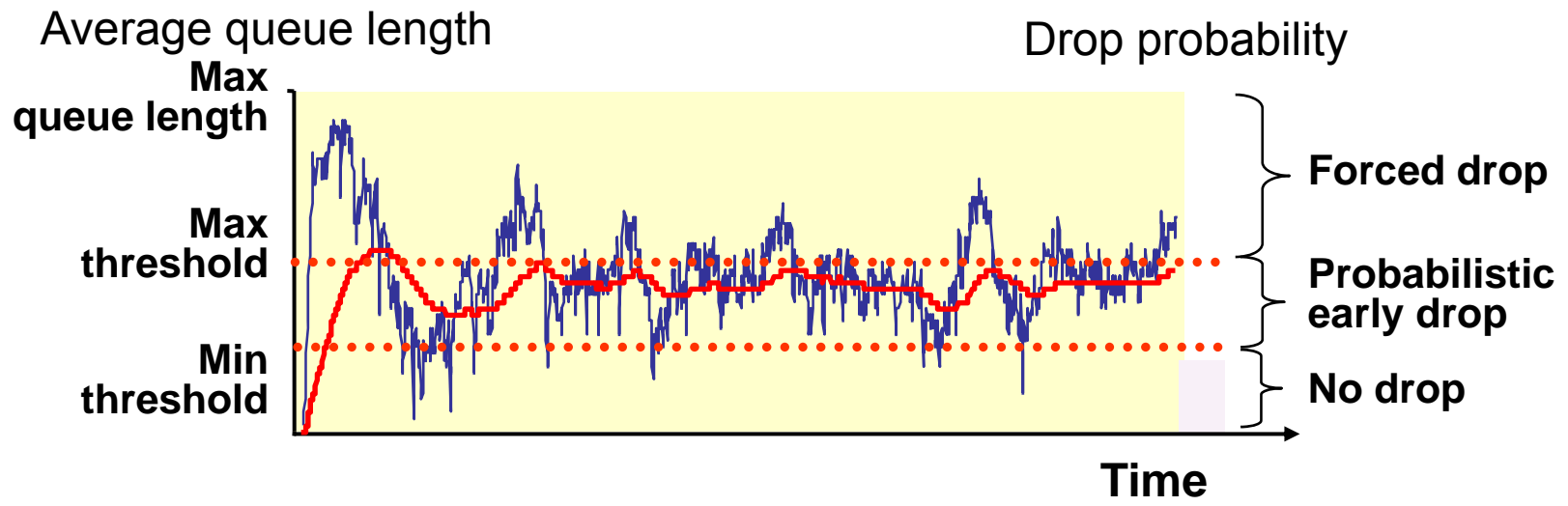
- ❑ probabilistic packet drop: flows see same *loss rate*
- ❑ problem: bursty traffic (burst arrives when queue is near threshold) can be over penalized

# 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



# 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

# 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 ...

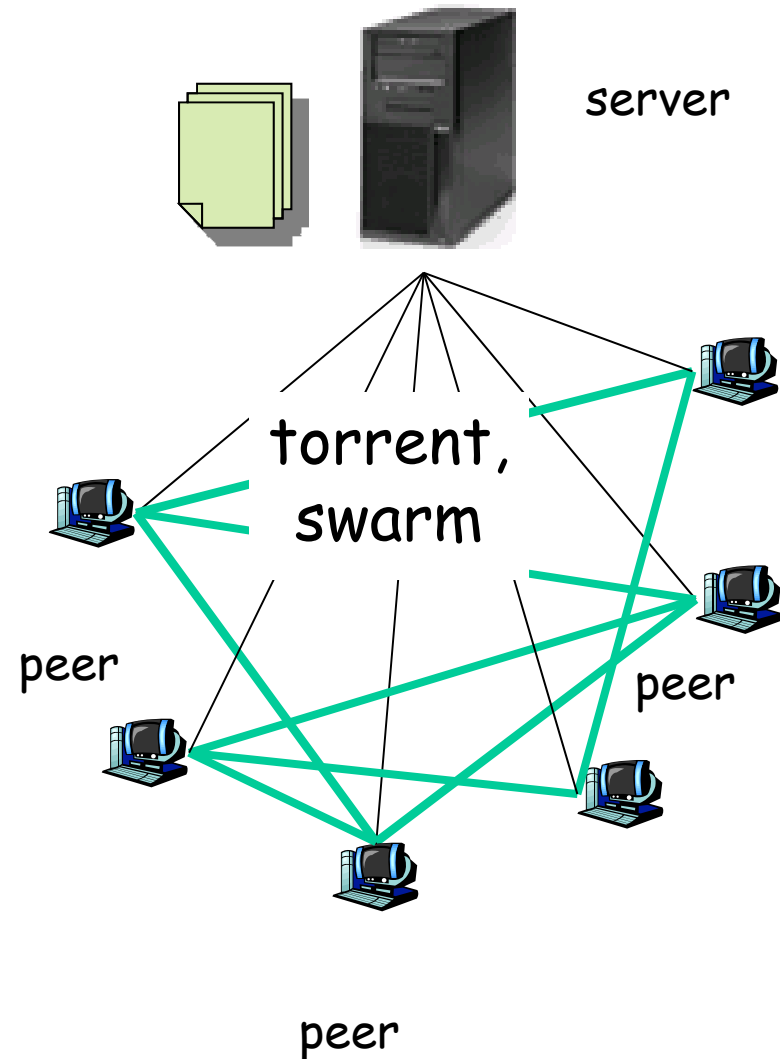
*We will revisit!*

# BitTorrent

- ❑ P2P protocol created ~ 2002
- ❑ popular: approximately 35% of Internet traffic (according to CacheLogic)

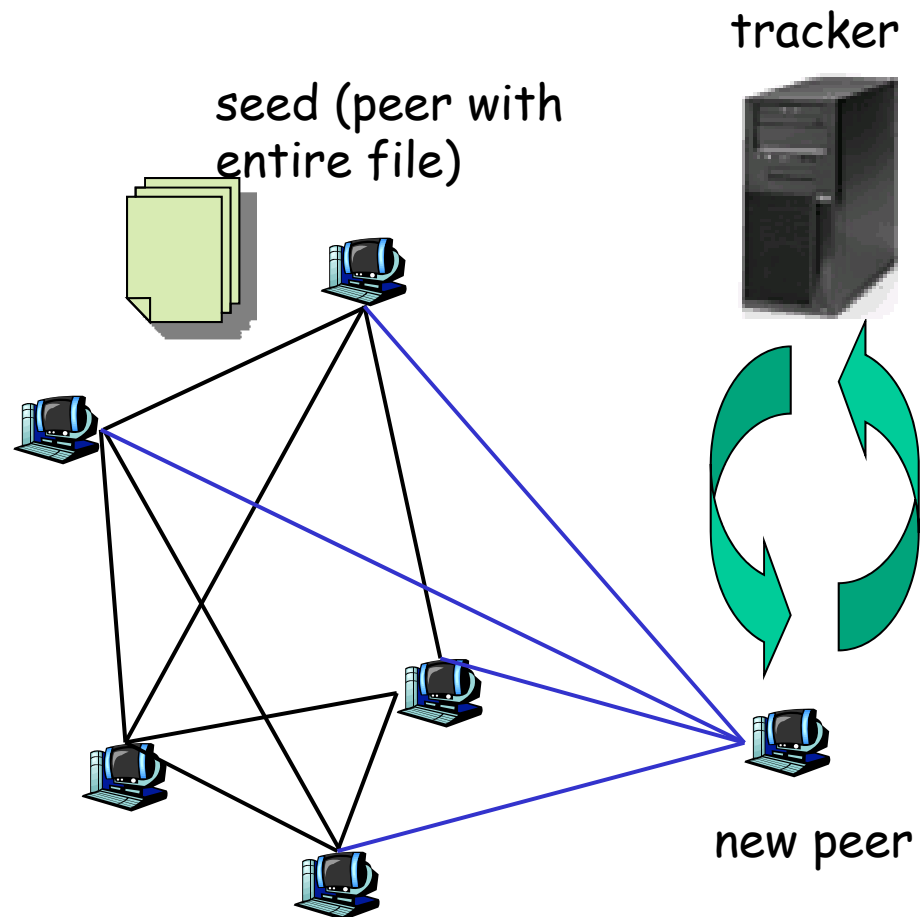
# BitTorrent Basics (Cohen)

- ❑ large file (~GB), large demand (10s, 1000s or more clients)
- ❑ divide file into small pieces (256KB).
- ❑ utilize all peers' upload capacities



# BitTorrent schematic

- new peer gets random list of peers (~50)
- chooses 4 peers randomly from list

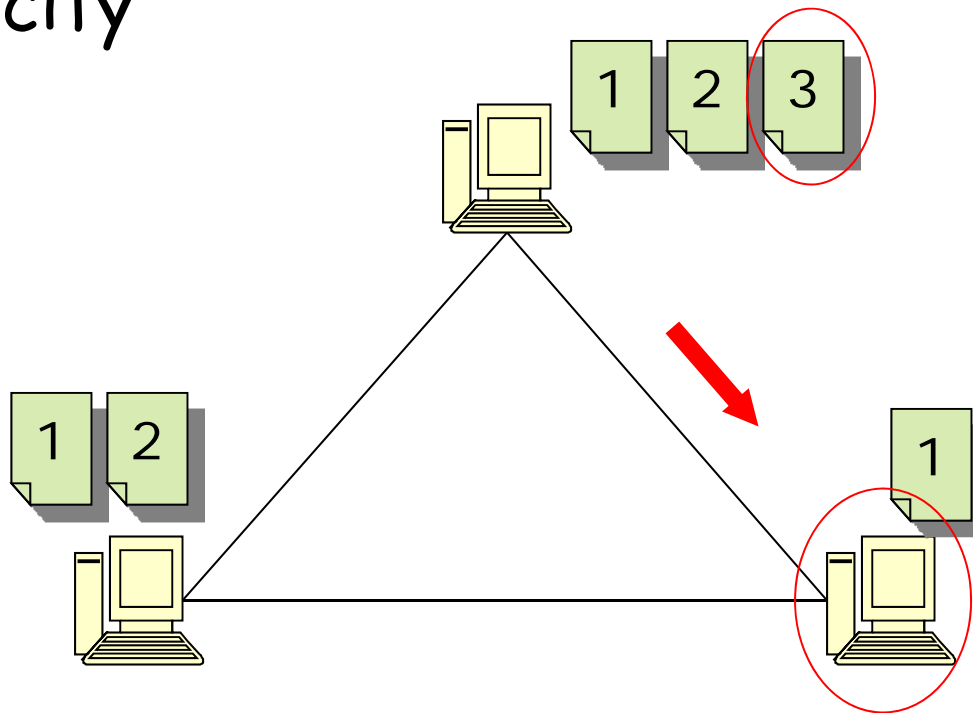


# Bittorrent: who to upload to?

- Tit-for-tat: upload to peers from which most data downloaded in last 30 seconds (4 peers by default)
- ⇒ incentive to upload in order to be chosen by other peers!

# Bittorrent : what piece to send?

- ❑ rarest-first: upload piece rarest among your neighbors first
- ❑ allows system to utilize full service capacity



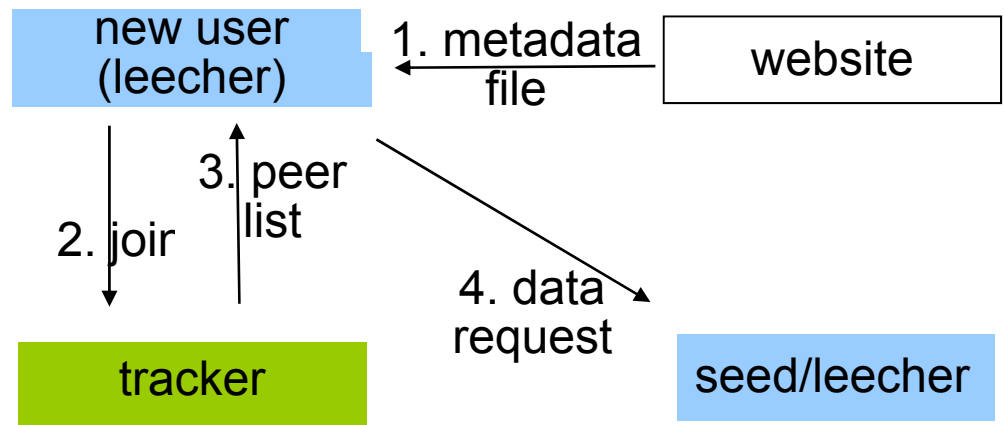
# Joining torrent

Peers divided into

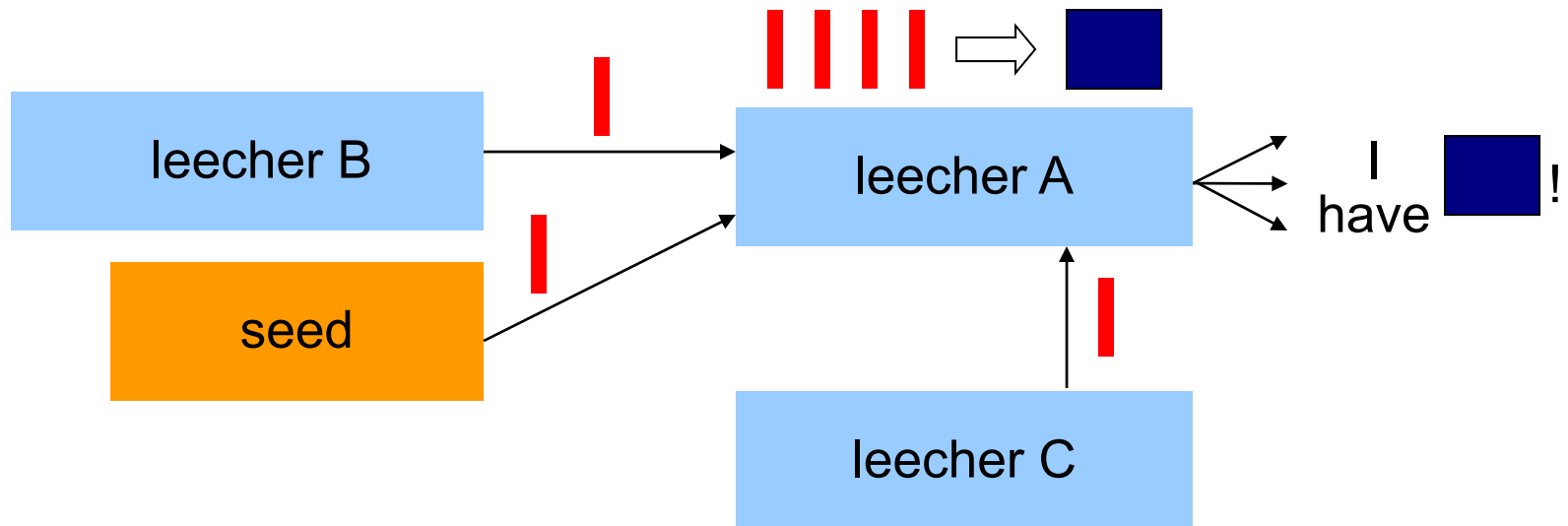
- ❑ *seeds* have entire file
- ❑ but *leechers* do not

Process:

- ❑ obtain metadata file (out of band)
- ❑ contact tracker
- ❑ obtain peer list (seeds, leechers)
- ❑ contact peers from peer list



# Exchanging data



- ❑ verify pieces  using hashes
- ❑ download sub-pieces (blocks)  in parallel
- ❑ advertise received pieces to peer list
- ❑ **interested**: need pieces that a given peer has

# Piece selection

- when downloading starts: choose at random
  - get complete pieces as quickly as possible
  - obtain something to trade
- after obtaining 4 pieces: (local) rarest first
  - achieves fastest replication of rare pieces
  - obtain something of value to trade
  - get unique pieces from seed
- endgame mode
  - defense against “last-block problem”
  - send requests for missing sub-pieces to all peers in our peer list
  - send cancel messages upon receipt of a sub-piece

# Last-block problem

- ❑ at end of the download, a peer may have trouble finding the missing pieces
- ❑ based on anecdotal evidence
- ❑ other proposals
  - network coding [Gkantsidis *et al.*, Infocom'05]
  - prefer to upload to peers with similar file completeness; unfair for the peers with most of the pieces [Tian *et al.*, Infocom'06]

# Peer selection

- ❑ four peers unchoked at a time
- ❑ every thirty seconds
  - drop peer with lowest download rate
  - every 30 sec. unchoke a random peer
- ❑ multi-purpose mechanism
  - allow bootstrapping of new clients
  - balances loads among peers -> lower delays
  - maintain connected network; every peer has non-zero chance of interacting with any other peer

# Peer selection

- ❑ four peers unchoked at a time
- ❑ every thirty seconds
  - drop peer with lowest download rate
  - every 30 sec. unchoke a random peer
- ❑ multi-purpose mechanism
  - allow bootstrapping of new clients
  - balances loads among peers -> lower delays
  - maintain connected network; every peer has non-zero chance of interacting with any other peer

# Power of two choices: load balancing

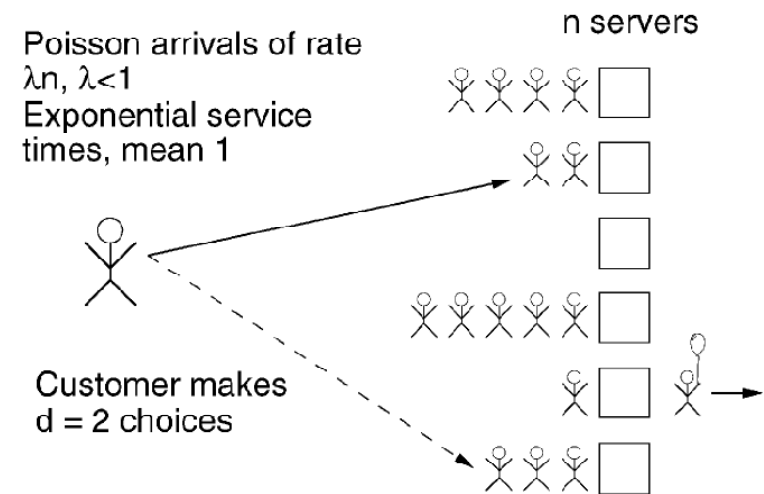
Load balancing problem:  
which of  $N$  queues to join?

## □ applications

- routing requests to web servers
- routing packets to queues/paths

## □ policies:

- random
- $SQ$ : join shortest queue: requires lots of info
- $SQ(d)$ : sample  $d$  queues at random, join shortest of  $d$  queues



The supermarket model, with  $d = 2$ .

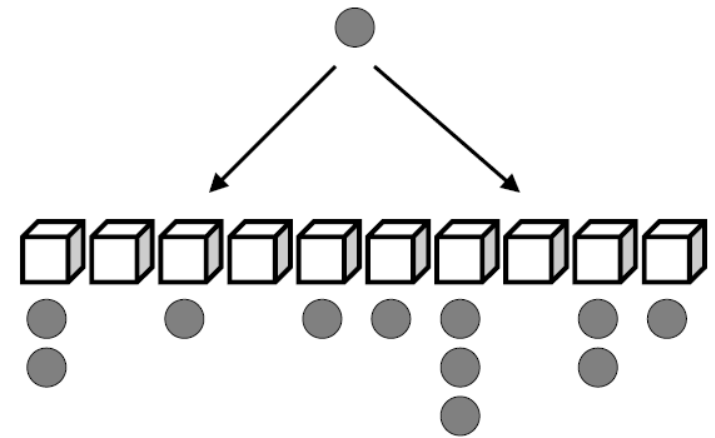
# The balls and bins problem

Place  $m$  balls in  $n$  bins

- Random:  $SQ(1)$
- $SQ(d)$ : sample  $d$  queues at random, place ball in bin with fewest balls

□ for  $m \approx n$ :

- $SQ(1)$ : max load is  $O(\log n)$
- $SQ(2)$ : max load is  $O(\log \log n)$
- $SQ(3)$ : only constant factor improvement over  $SQ(2)$



large improvement with small amount of info ( $d=2$ ), not much gain with additional info

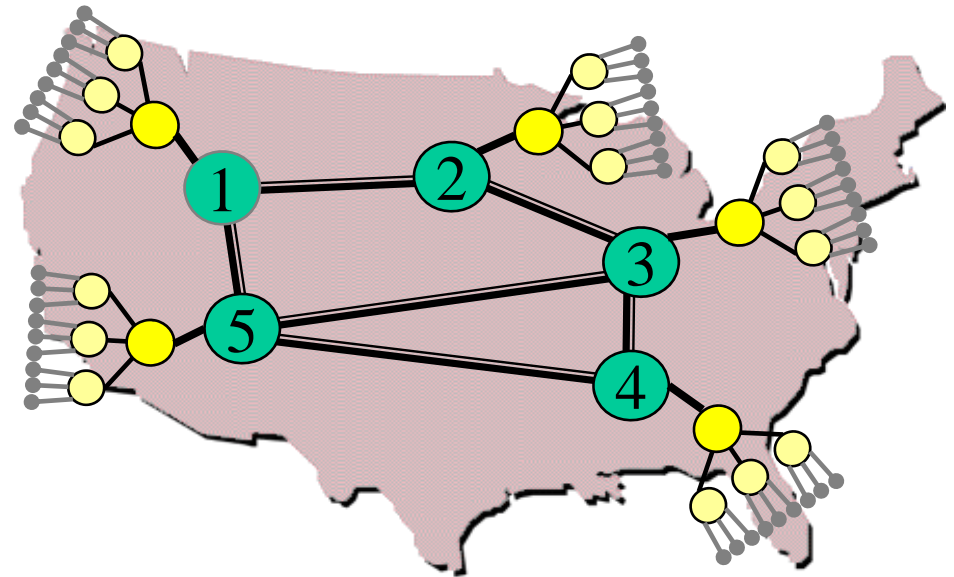
# The power of randomization

- *Rand*: easy to implement, performs poorly
- *SQ*: hard to implement performs well
- *SQ(2)*: choose randomly among two choices: easy to implement, performs exponentially better than RAND, getting "most" of benefits of SQ

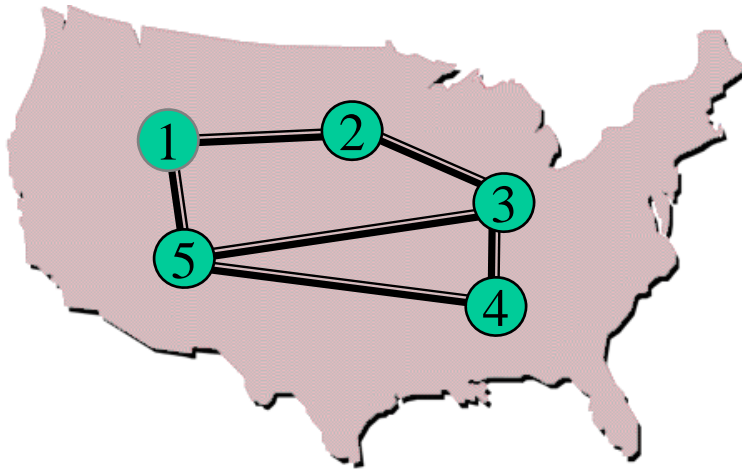
Ref: "The power of two choices in randomized load balancing,"  
M. Mitzenmacher, IEEE Trans. Parallel & Distributed Sys, Oct 2001

# Randomized two-hop routing

- *Goal*: determine path from ingress router to egress router
  - *robust* to changes in traffic:
    - links do not become overloaded, even as source/dest traffic rates change
  - *minimal overhead*
    - routing updates as traffic rates change



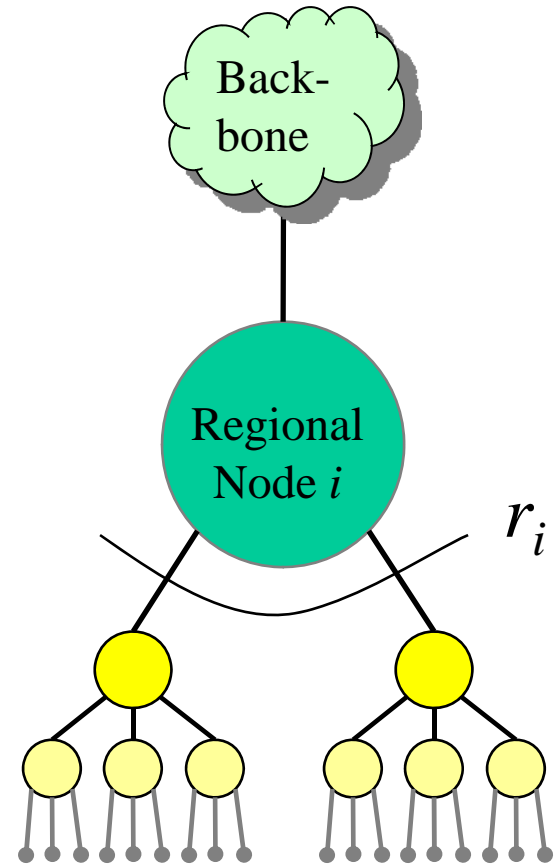
# Traffic Matrices



	To				
From	0	3	5	3	4
	2	0	8	1	2
	1	4	0	7	4
	8	2	1	0	5
	4	4	2	5	0

$r_i$

Traffic matrix hard to predict

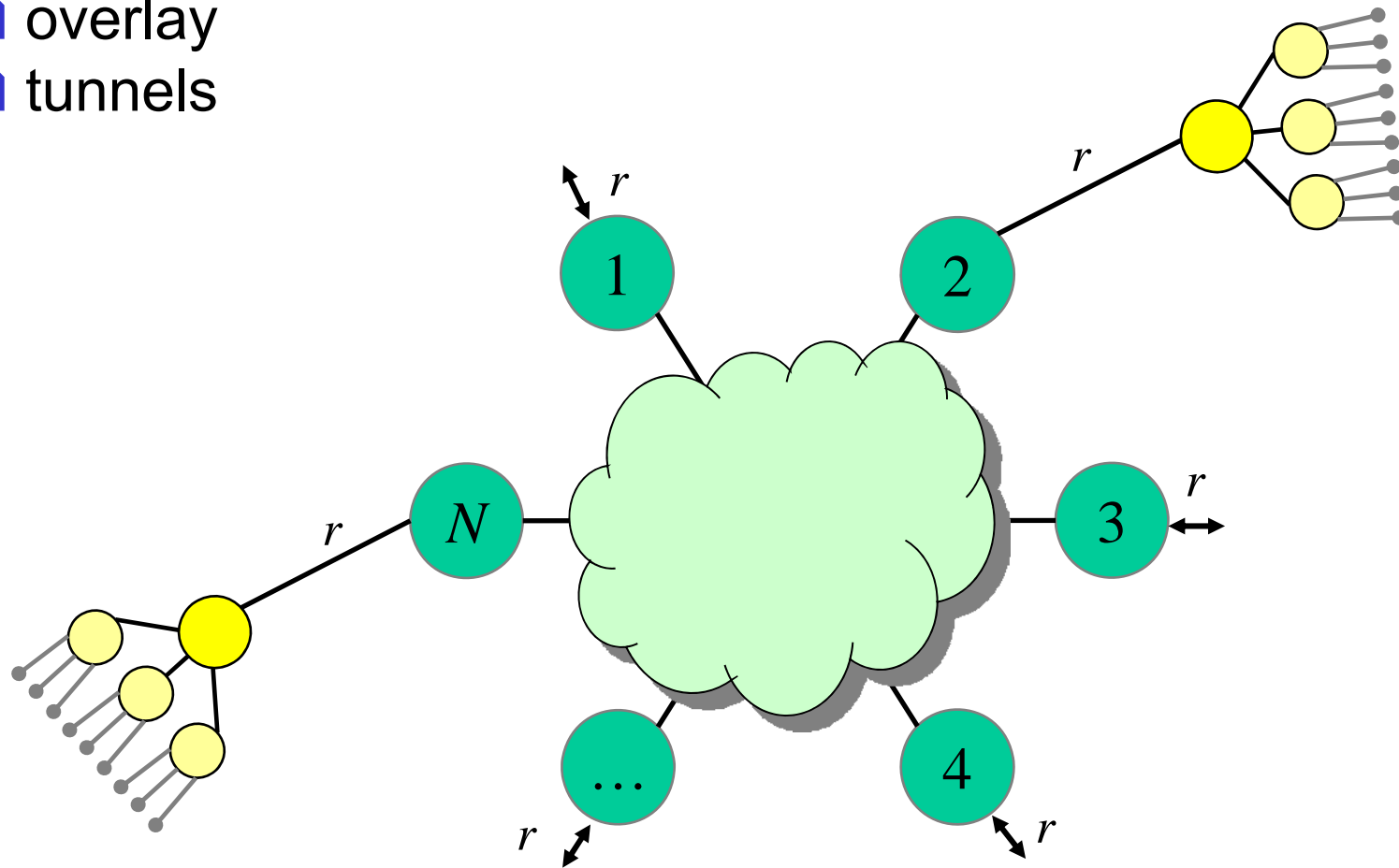


$r_i$  easier to predict  
and has to be predicted anyway

# Valiant Load-Balancing

Model: logical 1-hop connections between nodes

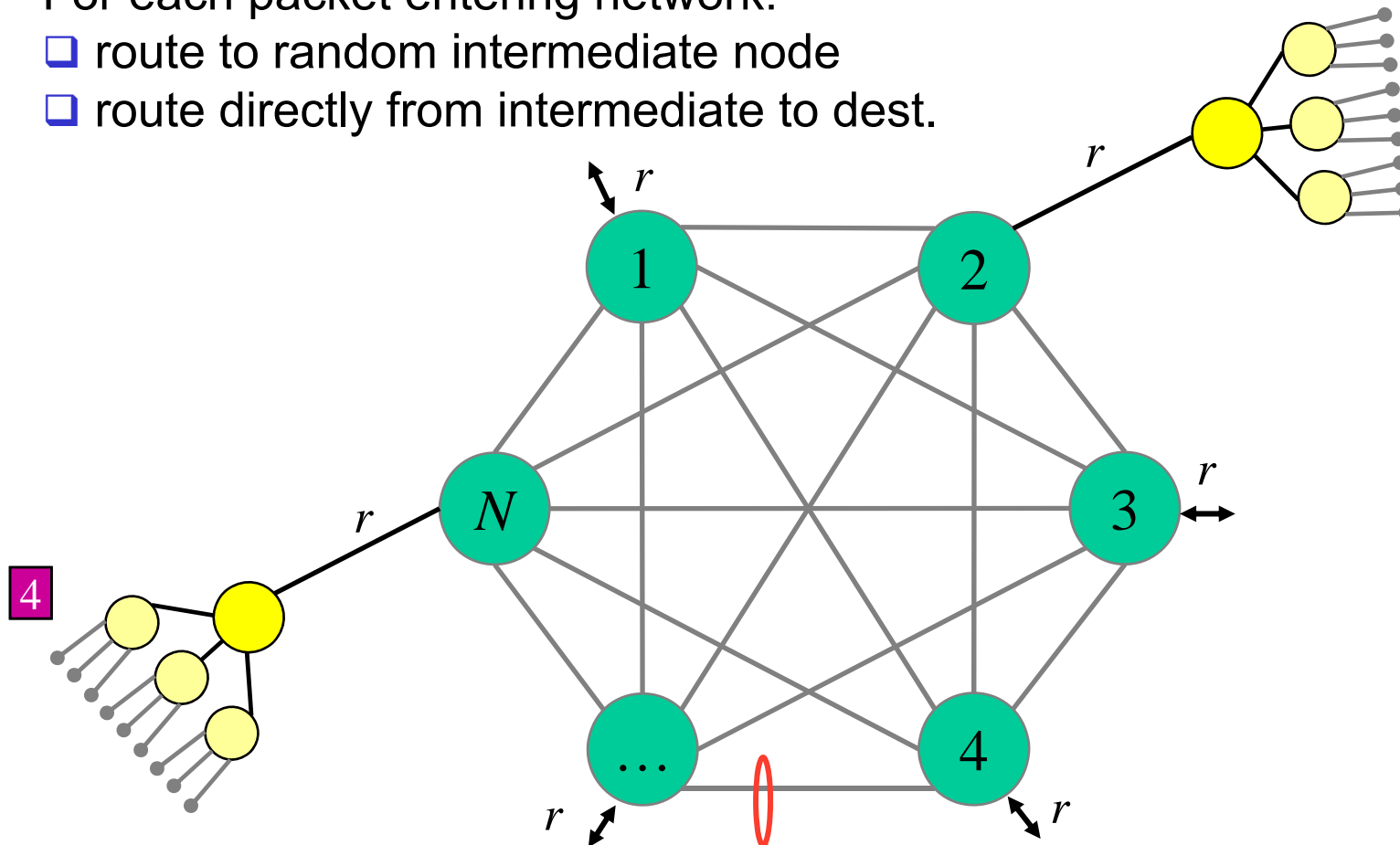
- overlay
- tunnels



# Valiant Load-Balancing

For each packet entering network:

- route to random intermediate node
- route directly from intermediate to dest.



**Q:** capacity needed for each link?

**A:**  $2r/N$

# Discussion

□ *seems inefficient: why not take direct one-hop route*

- 
- 

□ role of *randomization* here?

- 
- 

Ref: R. Zhang-Shen, N. McKeown, "Designing a Predictable Internet Backbone Network," HotNets III, San Diego, November 2004.