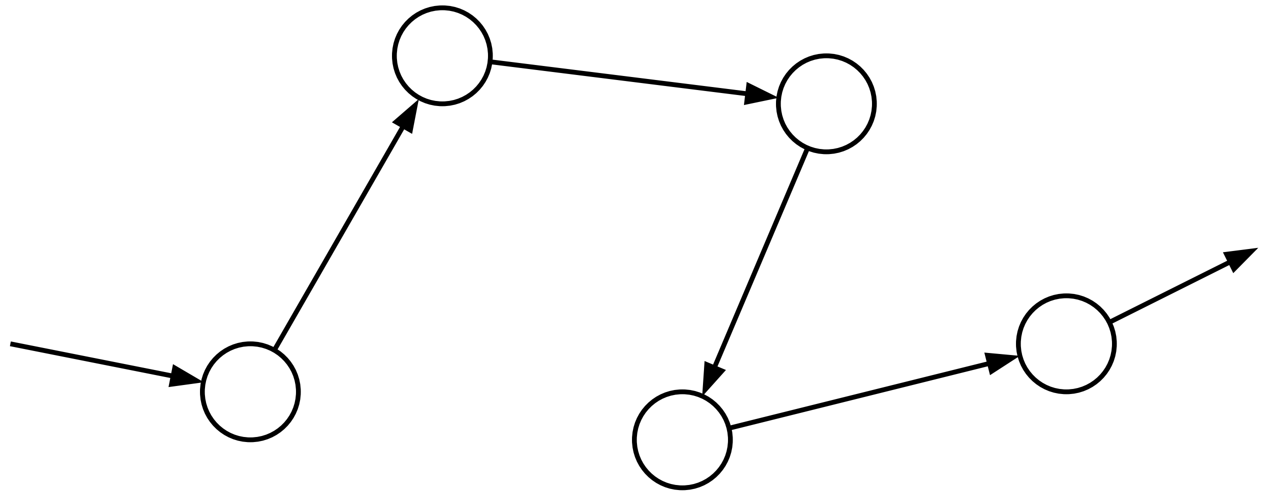
Robust Peer-to-peer Protocols via Randomized Forwarding

Adam M. Costello
University of California, Berkeley

2005-Dec-02-Fri

http://www.nicemice.net/amc/research/randomized-forwarding/

Introduction: forwarding



message (with destination field) relayed from node to node

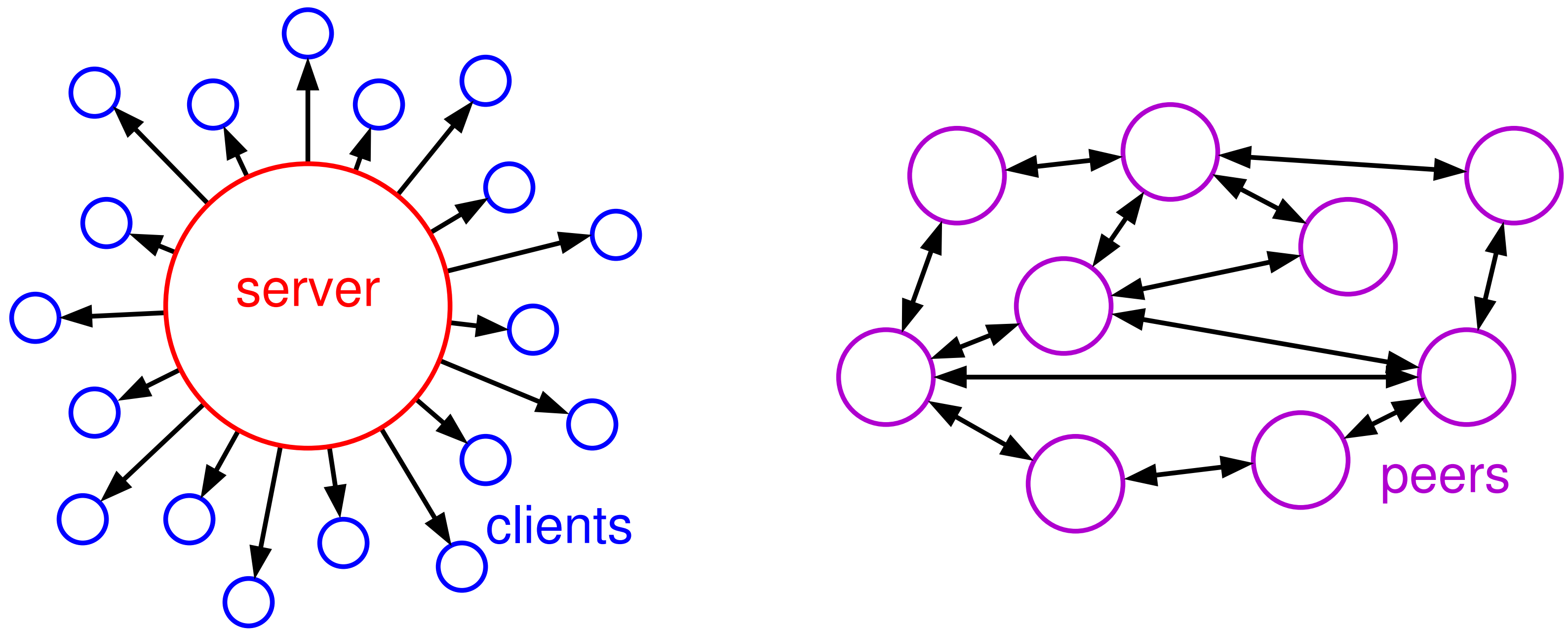
each node has a *routing table*:

destination $\Rightarrow$ next hop
$pattern_1 \Rightarrow node_1$
$pattern_2 \Rightarrow node_2$
$pattern_3 \Rightarrow node_3$

traditional focus is performance (shortest/fastest path)

faults must be detected, routing tables reconstructed

Introduction: peer-to-peer protocols & fault tolerance



pool resources ⇒ big reliable services from small unreliable servers

open system

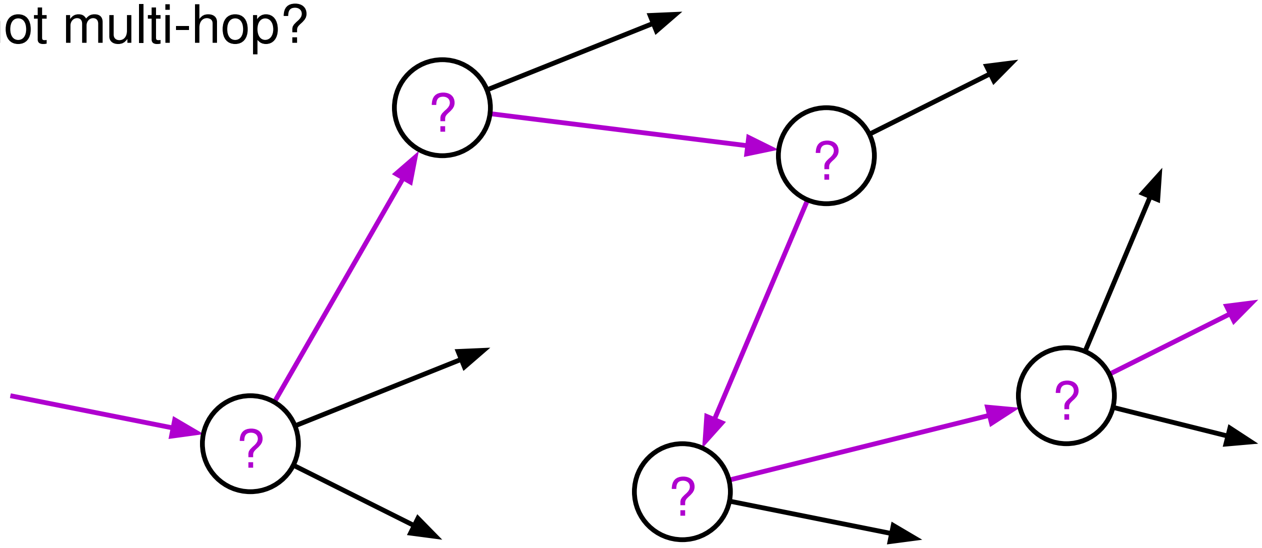
⇒ no one responsible for maintenance & repair

⇒ fault tolerance especially important

too many peers for anyone to know ⇒ multi-hop message routing

Introduction: randomized forwarding

randomization works for one-hop communication (gossip protocols), why not multi-hop?



destination $\Rightarrow$ next hop (choose one at random)
$\mathit{pattern}_1 \Rightarrow \mathit{node}_{10}$ or node_{14} or $\ldots$
$\mathit{pattern}_2 \Rightarrow \mathit{node}_{23}$ or node_{26} or $\ldots$

original sender (only) resends message until at least one succeeds, each copy takes a different path

rather than detect & route around faults, preemptively route around *everything*

Outline

multicast loss recovery

- randomized forwarding at network layer:
- Search Party
- Rumor Mill (briefly)

distributed hash tables

- randomized forwarding at application layer:
- Tangle

Outline

multicast loss recovery

- introduction
- LMS (recovery using parentcast)
- Search Party (recovery using randomcast)
- methodology
- results
- Rumor Mill (simpler, at some cost)

distributed hash tables

Multicast packet loss

network builds a
multicast distribution tree

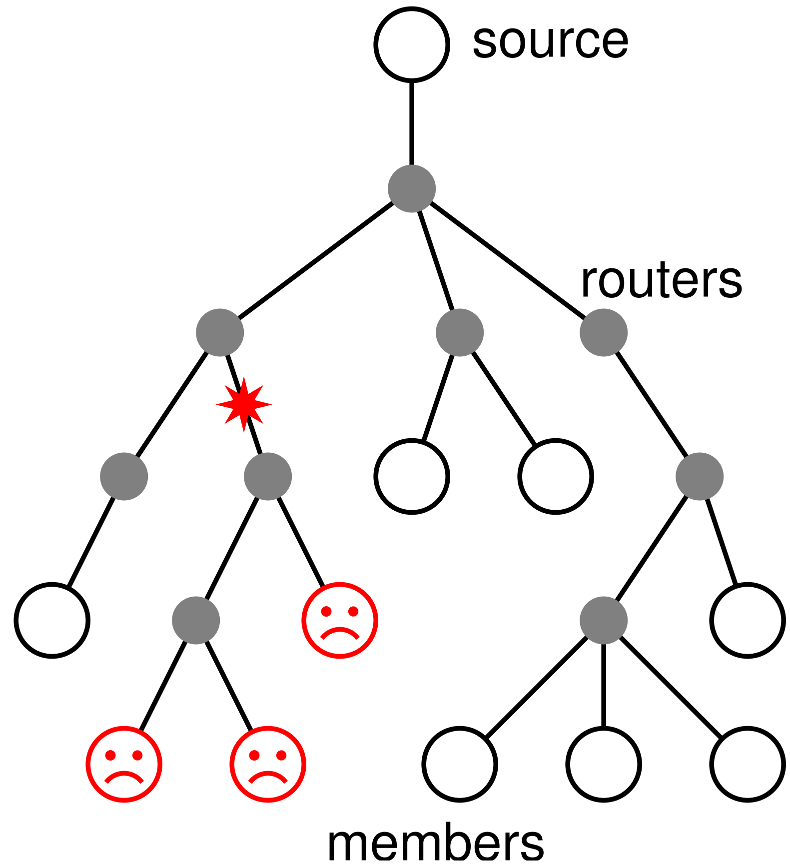
packet loss affects members of the
loss subtree

unicast repairs from source?
  does not scale

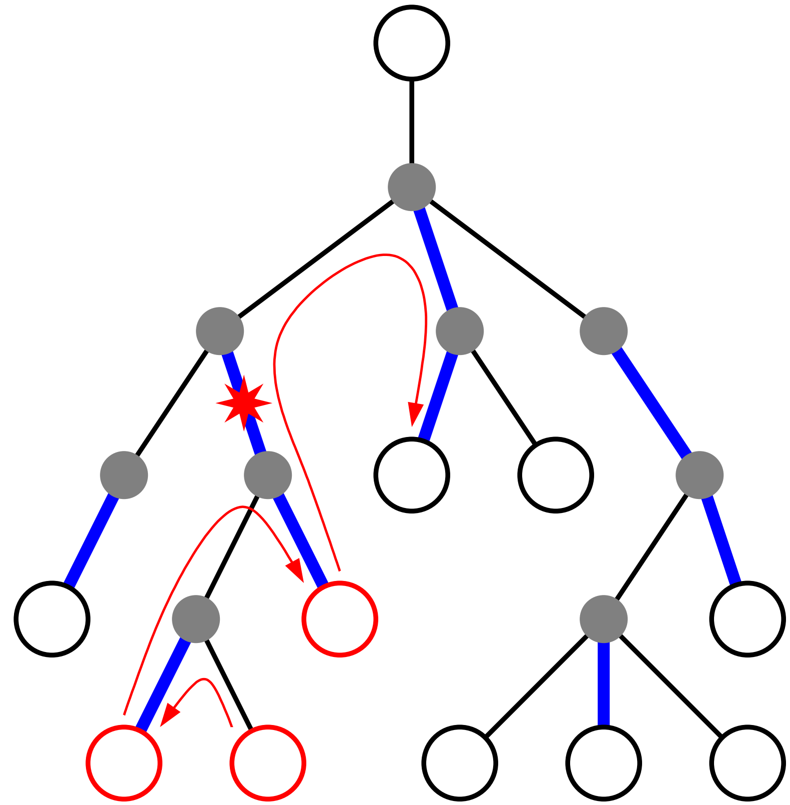
multicast repairs?
  does not scale

per-packet state in routers?
  layer violation

members form a hierarchy?
  efficient only if aligned with network topology ⇒
  layer violation



LMS [PPV98]



network builds the hierarchy: one edge flagged below each router

parentcast: forward up until unflagged edge is crossed (*turning point*, recorded in packet), then down along flagged edges

exactly one parentcast request escapes the loss subtree

repair is *subcast* to all members below turning point

simple, elegant, fast, efficient, no layer violations, but...

entire loss subtree depends on just two members

repair subtree can be much larger than loss subtree (*exposure*)

Search Party

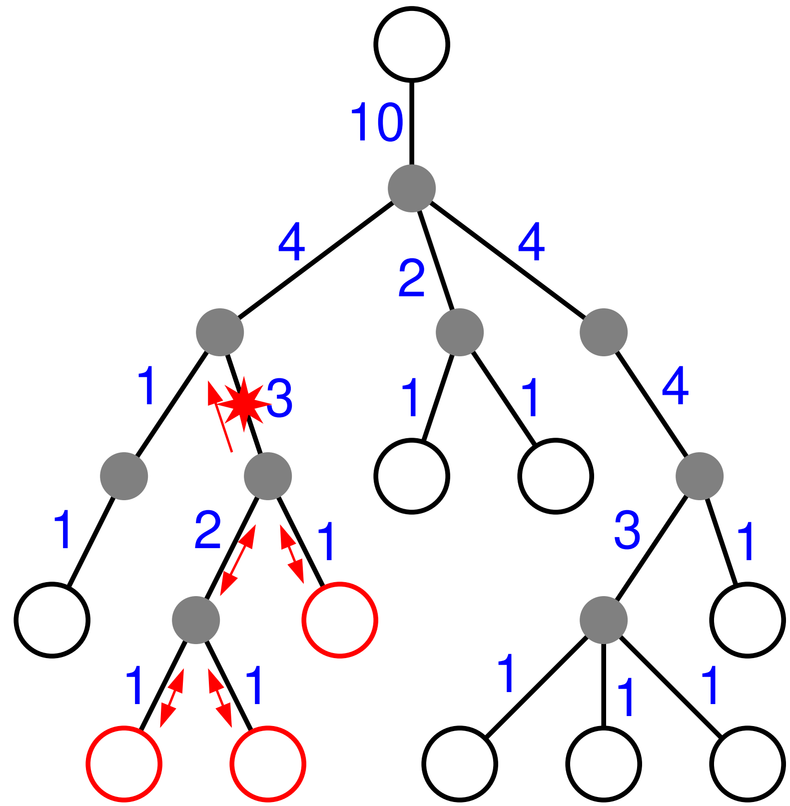
edges tagged with number of members below

randomcast: continue up with probability tag(in)/tag(up) else go down any edge uniformly (but no backtrack)

each member randomcasts requests at rate $\mu \Rightarrow$ requests escape loss subtree at rate μ

escaped request equally likely from any member of loss subtree, many likely destinations

repair is subcast as in LMS



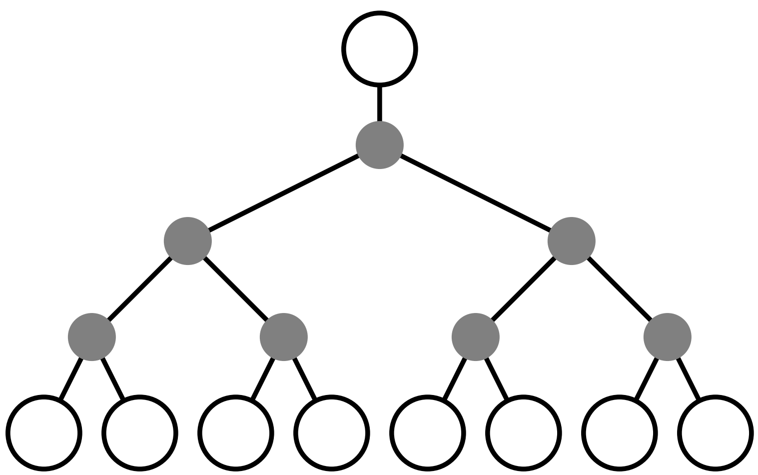
Methodology

mathematical analysis (for arbitrary topologies):

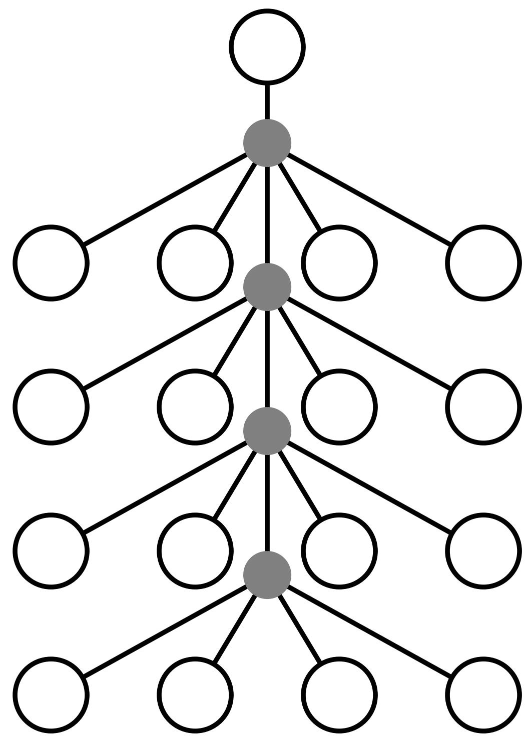
- proven cost bounds
- performance approximations

network simulation using **ns**:

- small-scale (up to 64 nodes)
- benchmark topologies



balanced



square

Analytic results

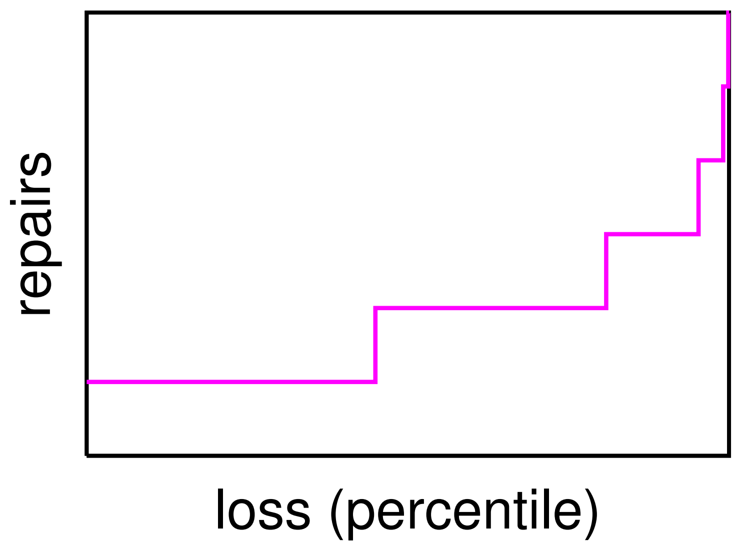
proved: if all L members of loss subtree randomcast requests at rate $\mu\ldots$

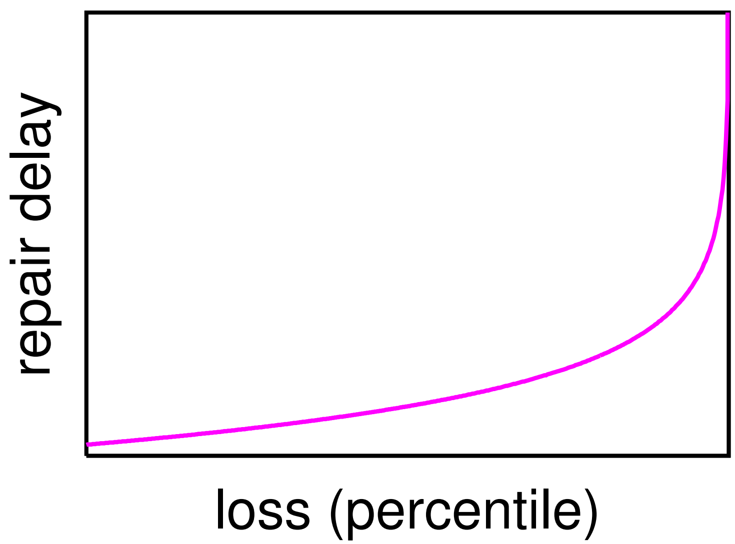
downward traffic on any link $\leq 2\mu$ (LMS: up to $height \times [fanout - 1]$)

average exposure $\leq 1 + \ln(N/L)$ (LMS: could be N/L)

approximated: for nominal round-trip time r between members$\ldots$

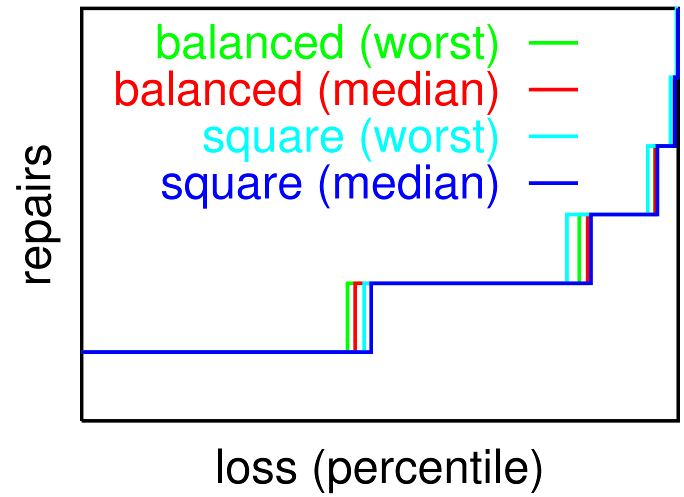
repairs$/$loss $= 1+$ Poisson$(r\mu)$ repair delay $= r+$ exponential$(1/\mu)$



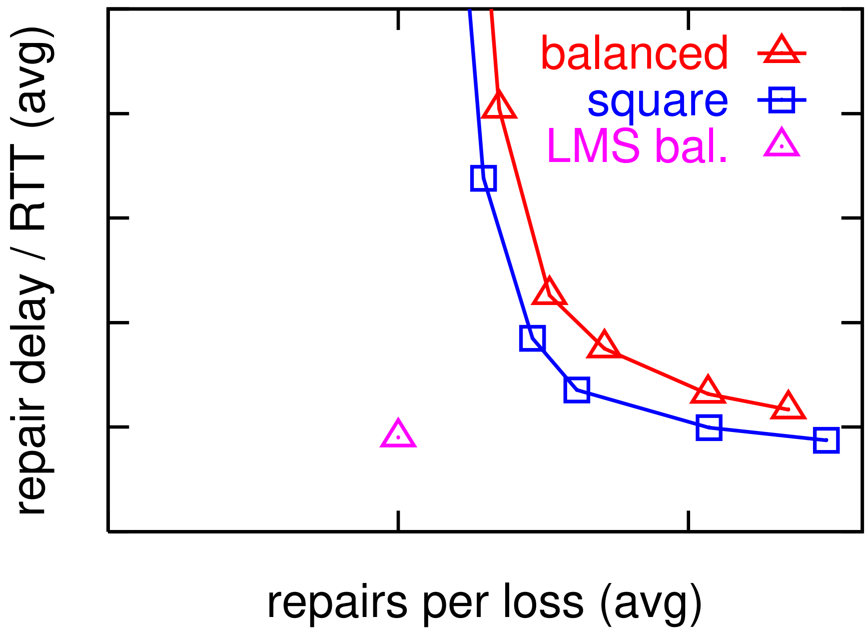


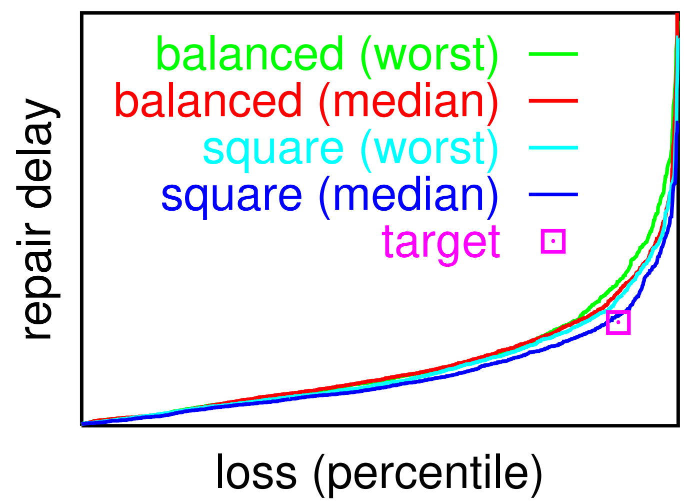
Simulation results

approximations match reality well enough for nodes to measure r and set μ to meet delay & duplication targets:



tradeoff between delay & duplication, can get both within 2x of LMS:





Rumor Mill

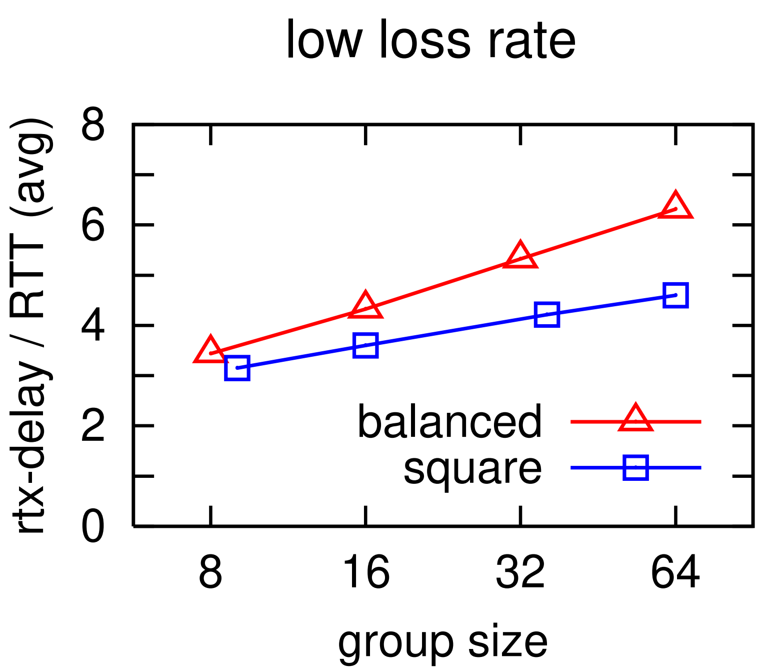
repairs are unicast, not subcast

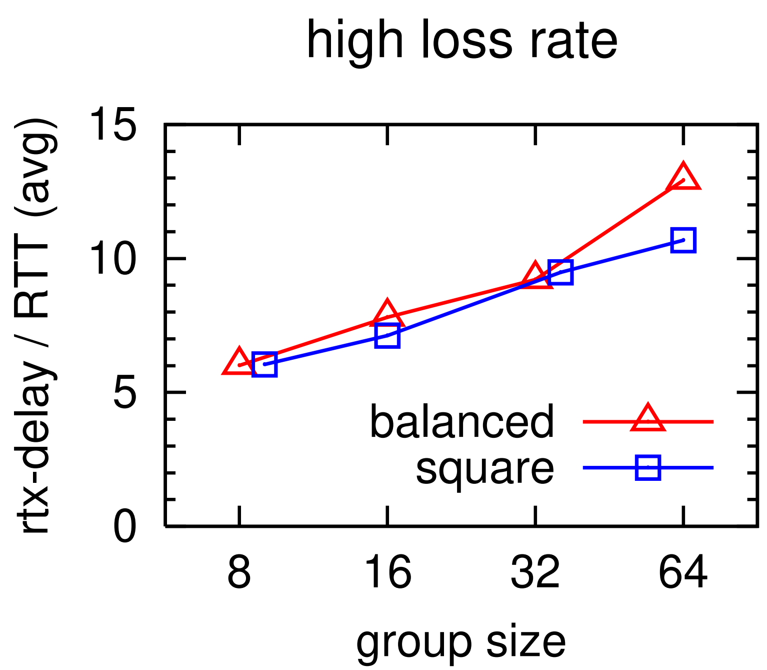
requests use *trivial randomcast*:

- uniform random walk ⇒ no router state (no edge tags)
- no turning point ⇒ no packet modification

requests per packet adapts to loss frequency

easier to deploy, but repairs spread more slowly, like a rumor





Multicast loss recovery summary

getting repairs from the source is problematic

LMS introduced a good architecture:

- peer-to-peer loss recovery built on new forwarding services
- scalable, fast, efficient
- but too sensitive to node failures and location of losses

Search Party:

- randomized forwarding $\Rightarrow$
  - proven diffusion of responsibility
  - proven elimination of sensitivity to loss location
  - modest overhead (analytic approximation corroborated by simulation)

Rumor Mill:

- much simpler, but slower by $O(\log N)$

Outline

multicast loss recovery

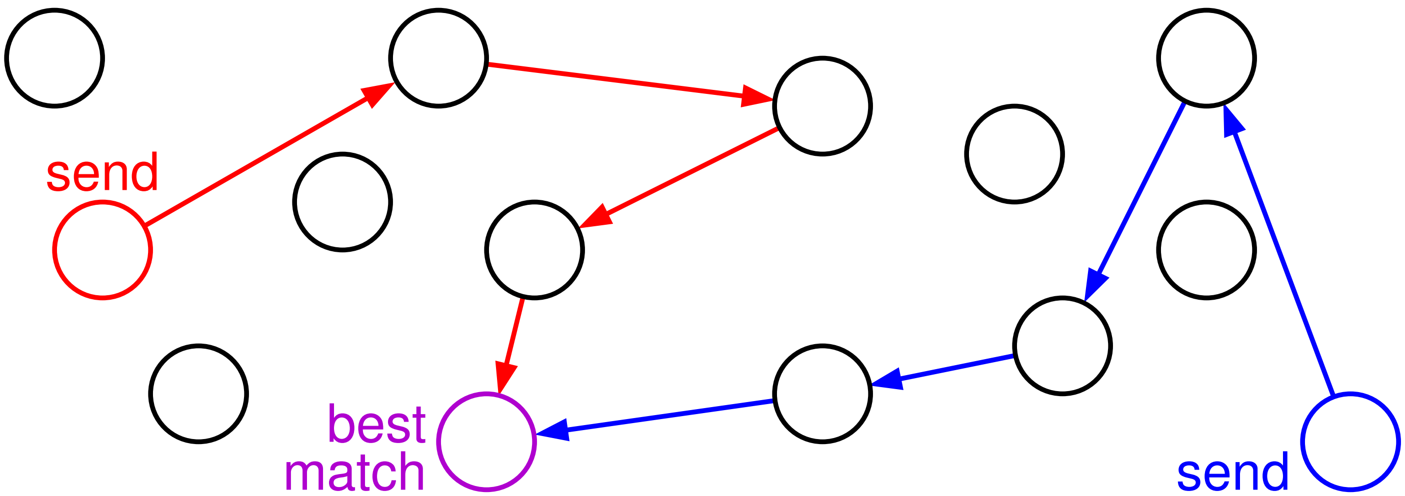
distributed hash tables
- introduction
- Tangle, a randomized DHT
- methodology
- results

Distributed hash tables

nodes are assigned *labels* uniformly from an abstract space

send(label,message) forwards message to node with best matching label

useful for rendezvous
(example: “put” and “get” messages for a hash table)



protocols: CAN, Chord, Pastry, Tapestry, Tangle, . . .
(any could be modified to use randomized forwarding like Tangle does)

Tangle overview

put(label,version,value) adds (version,value) to set
get(label) returns random (version,value) pairs from set

hash table layer	soft-state redundant storage, implosion prevention, newer-version notification; at each hop store/retrieve data, send response, and/or terminate forwarding

send(label,message)

routing layer	forwarding sublayer (route selection): choose next hop for relaying messages
	peer discovery sublayer (neighbor selection): construct soft-state peer info (routing table)

Tangle routing table

node label = hash of network address (bit string)

[PRR97] proximity metric:
goodness of match = length of common prefix

example: my label = 101101000. . .

prefix length	pattern	next hop
0	0∗	[random pool of nodes]
1	11∗	[random pool of nodes]
2	100∗	[random pool of nodes]
3	1010∗	[random pool of nodes]
4	10111∗	[random pool of nodes]
	etc.	

example: target label = 101001110. . .
common prefix = 101, use row 3

hop improves match by at least one bit (sometimes more)

Tangle forwarding

next hop: select any peer in the appropriate pool (if not empty)

message contains a *forwarding hint* to control selection method:

- deterministic $\Rightarrow$ fastest peer (max prefixGrowth / RTT)
- random, uniform $\Rightarrow$ all peers in row equally likely
- random, skewed for speed $\Rightarrow$ favor greater prefixGrowth / RTT
- random, skewed for hop-size $\Rightarrow$ favor greater prefixGrowth

original sender (only) resends until success:

deterministic send() $\Rightarrow$ all messages deterministic
randomized send() $\Rightarrow$ all messages random
hybrid send() $\Rightarrow$ first transmission deterministic, retransmissions random

Tangle peer discovery

inspired by gossip protocol Name-Dropper [HLL99]

continual exchange of random peer addresses with random peers

prefix length	pattern	next hop	rumored
0	$0*$	[pool of nodes]	[pool of nodes]
1	$11*$	[pool of nodes]	[pool of nodes]
2	$100*$	[pool of nodes]	[pool of nodes]
3	$1010*$	[pool of nodes]	[pool of nodes]

delete a rumored node, attempt a ping-pong-pong handshake with it

pong adds peer to next-hop column, contains a few of peer's next hop nodes (chosen non-uniformly at random) to add to rumored column

careful non-uniform choice $\Rightarrow$ all rows refreshed at same rate (see spec for details)

if a pool exceeds size limit, delete a random member

discard next hop nodes when too old

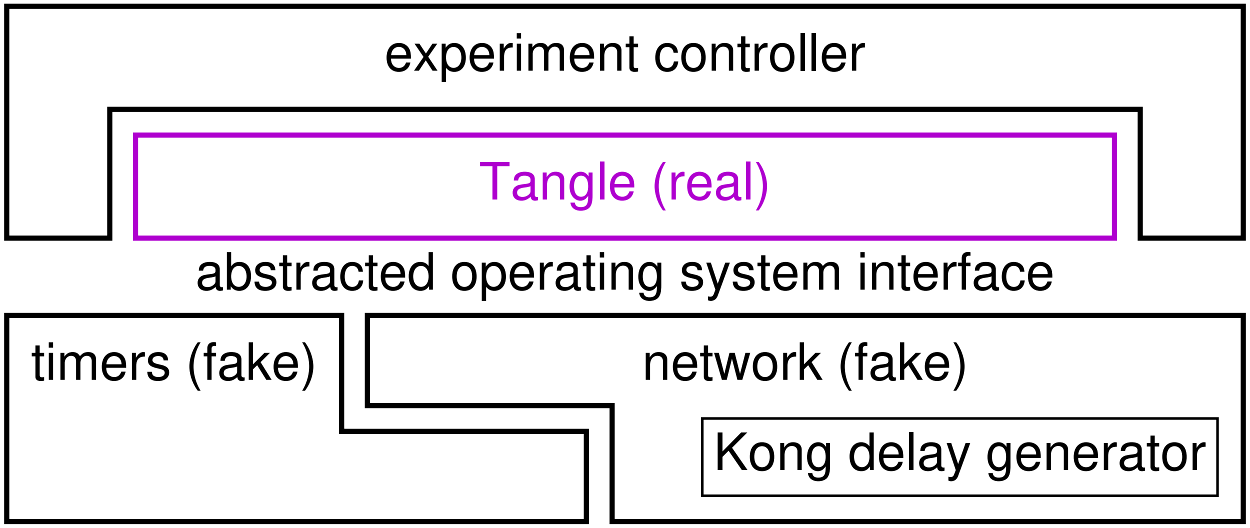
Outline

multicast loss recovery

distributed hash tables

- introduction
- Tangle, a randomized DHT
- methodology
  - experimental framework
    - network delay generator (Kong)
  - experiments
- results

Experimental framework



864 MHz Pentium III, 512 MB RAM

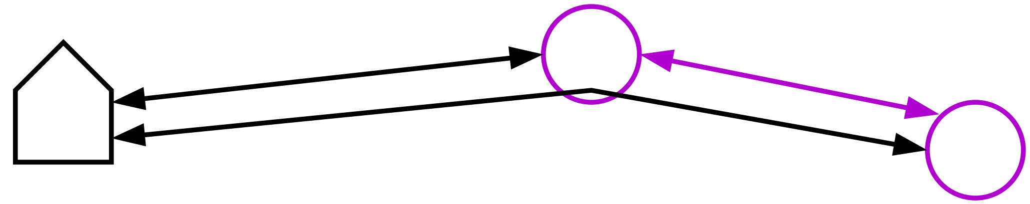
32768 nodes run in single thread at about 40% real-time

connectionless intermittent low-bandwidth communication

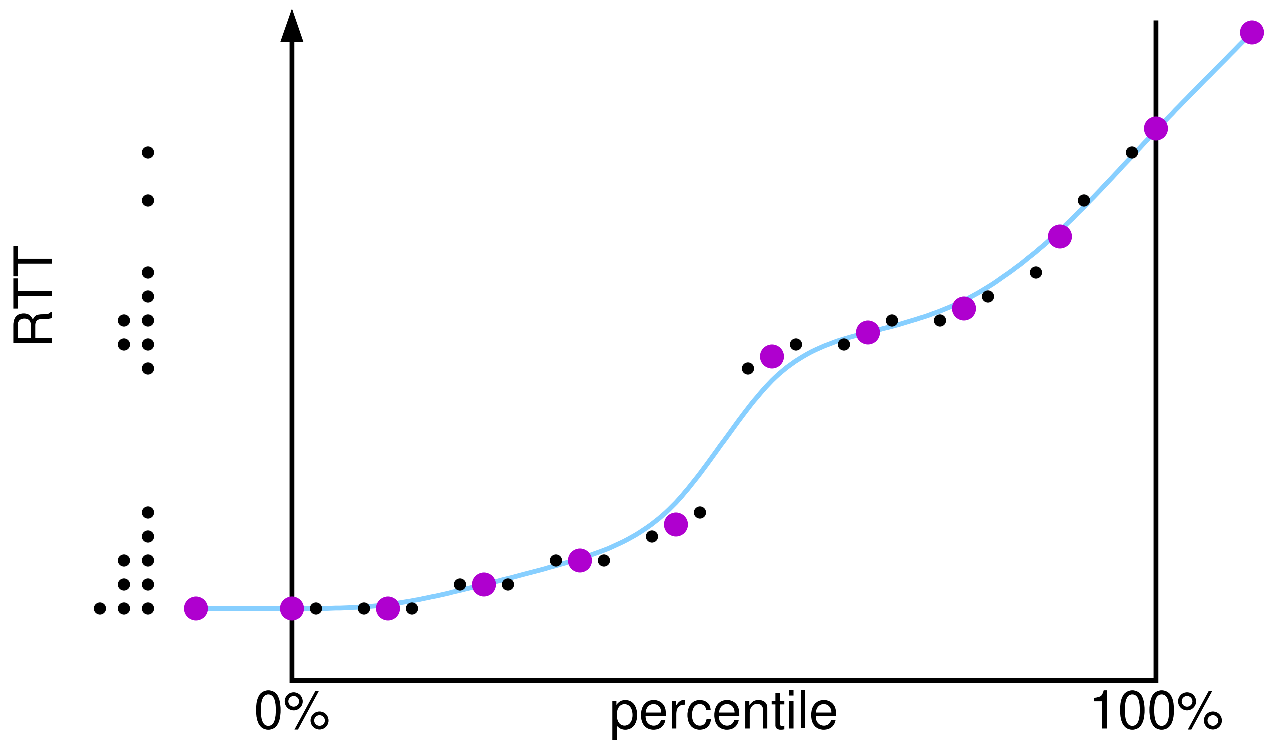
$\Rightarrow$ no need to simulate topology, only end-to-end delays

Kong: The network delay ape (overview)

1. measure round-trip-times between host pairs as in **king** [GSG02]

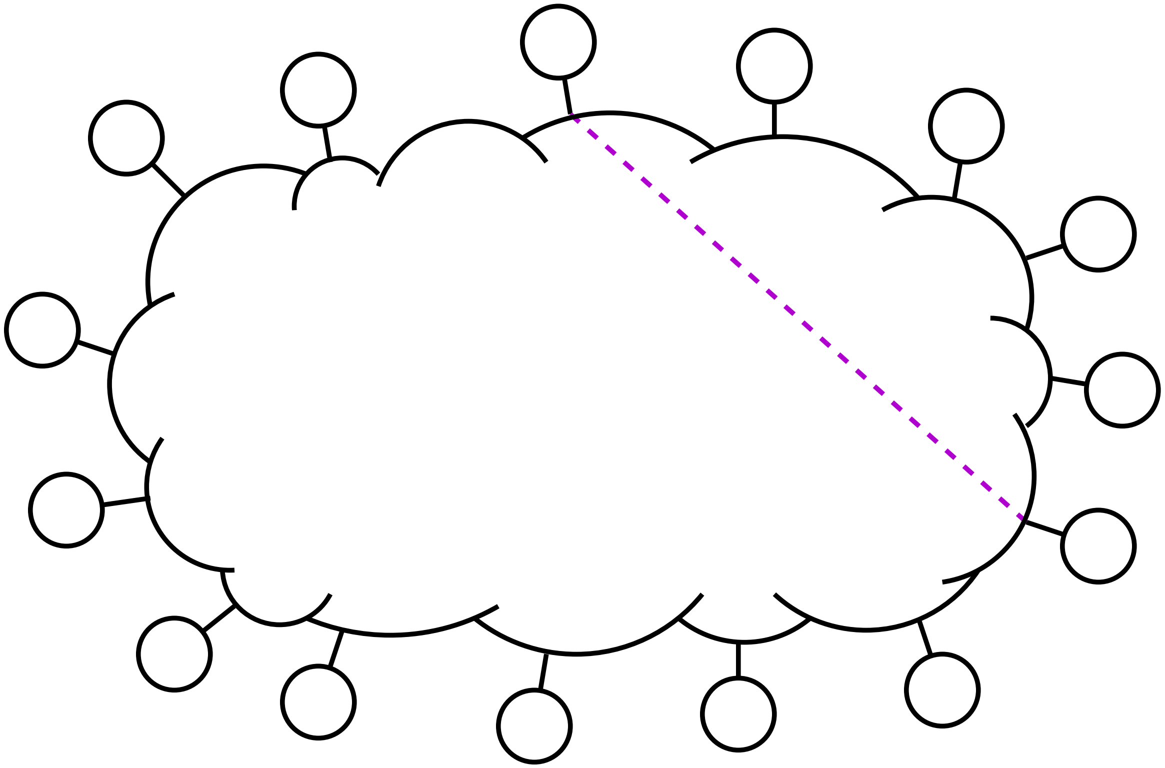


2. approximate each pair's RTT distribution (sort, filter, subsample)



3. mimic the original random variable using cubic interpolation

Kong core



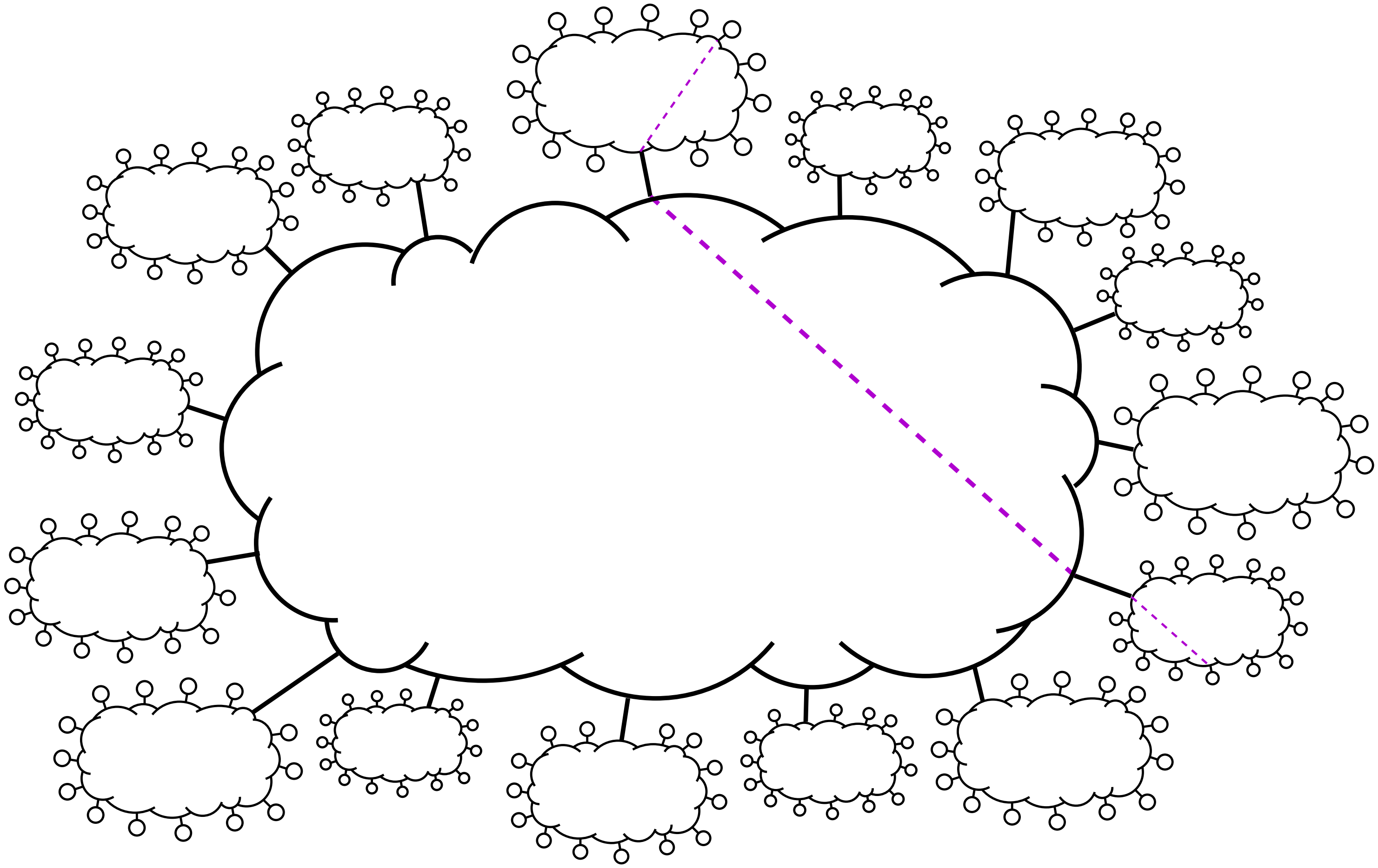
386 endpoints $\Rightarrow$ 74,305 pairs $\Rightarrow$ 891,660 floats

n endpoints $\Rightarrow O(n^2)$ space & $O(n^2)$ measurement time

$\Rightarrow$ limiting factor of whole simulation : (

Kong scaling

Let each node represent a local area (scaled-down copy of wide area):



n endpoints $\Rightarrow O(n)$ space & $O(n)$ measurement time

(bogus model, but local areas are small $\Rightarrow$ effect is small)

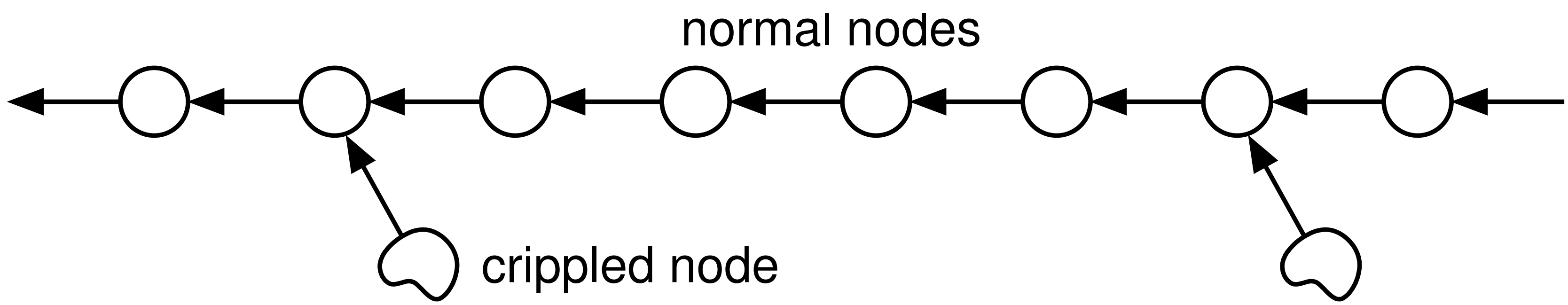
Experimental setup

measure routing layer send() response time & failure rate
send() retransmits twice per second until response or 20 second limit

upper hash table layer not used

32768 nodes

initial routing tables form chain with no network locality



steady-state faults: evolve routing tables with fault, freeze, measure
dynamic faults: measure before & after instantaneous fault (no freeze)

allow 4000 seconds to reach equilibrium (3x–5x longer than needed)

aggregate 20 independent runs

send() styles include deterministic and hybrid

Experiments

steady-state faults	no faults packet loss (in network, at receivers, at senders) route loss (invisible peers, blind peers) bugs (peers that cannot forward) membership churn (transient peers)
dynamic faults	membership burst (mass join, mass leave)
scaling	settling time (time to construct routing tables) response time (mean, in equilibrium)

Results

deterministic send()

performs well (fast response time)

except when it completely fails

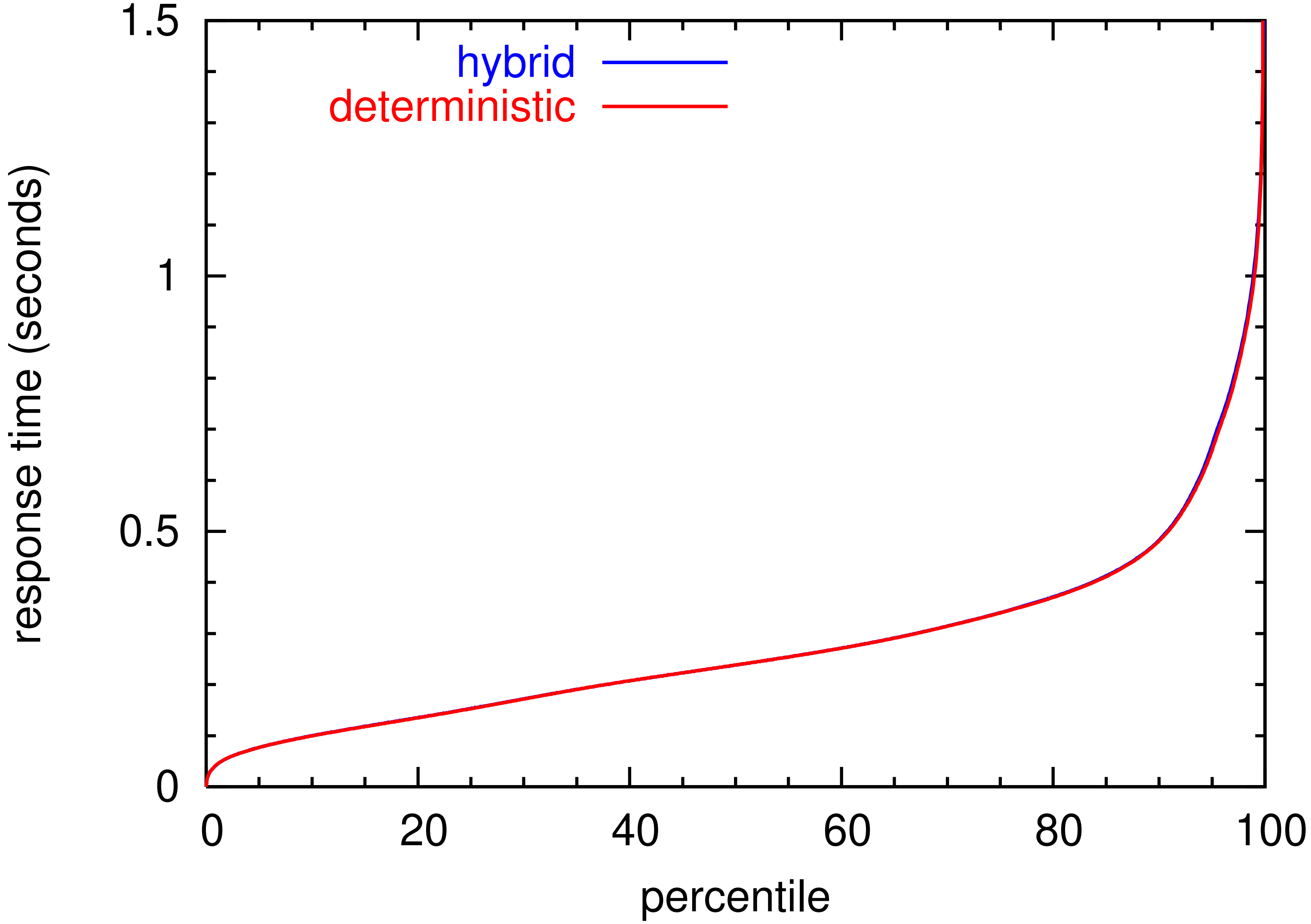
hybrid send()

mostly preserves good performance

greatly reduces failure rate

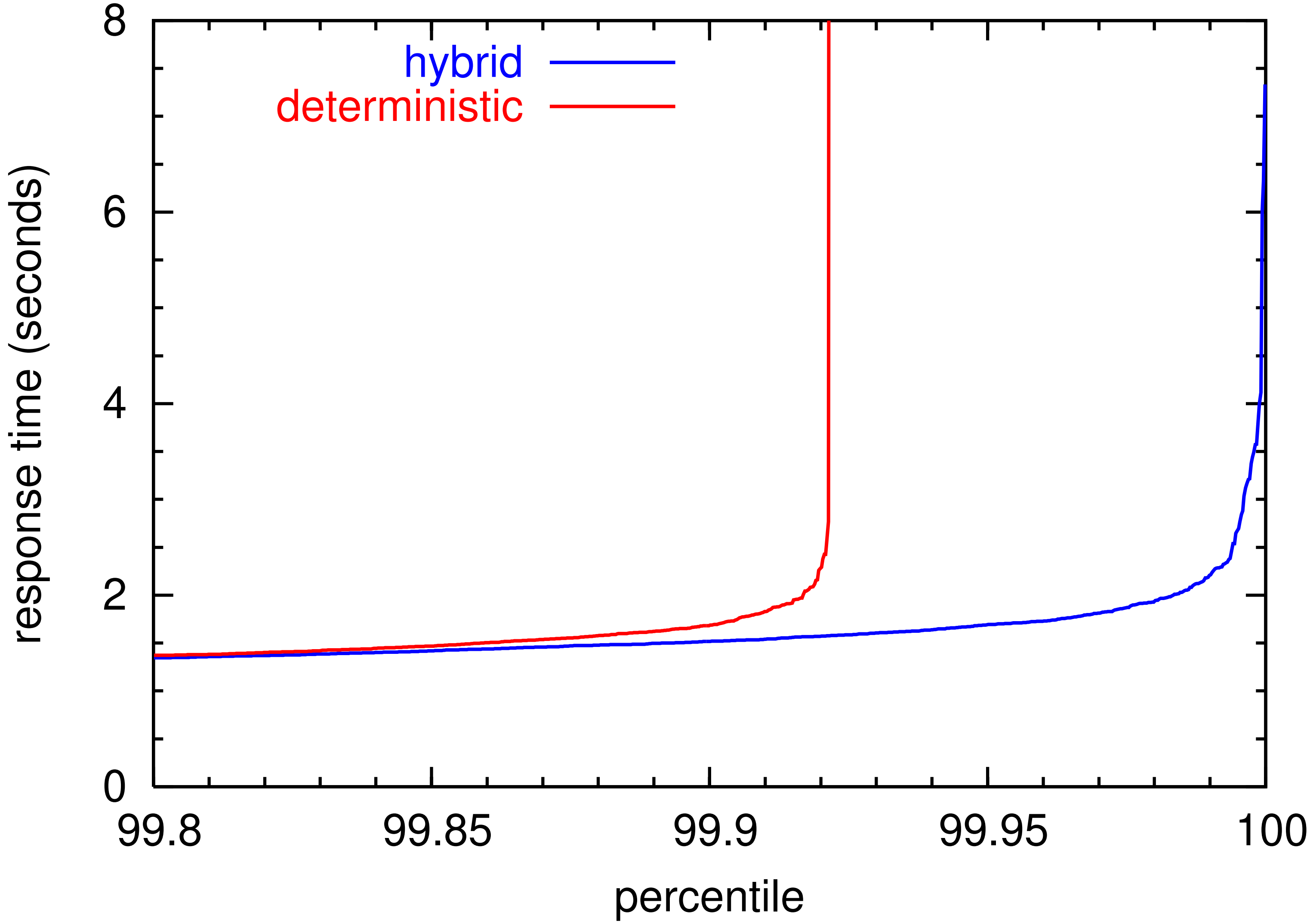
Results (no faults, response time)

without faults, no cost for hybrid randomization



Results (no faults, failure rate)

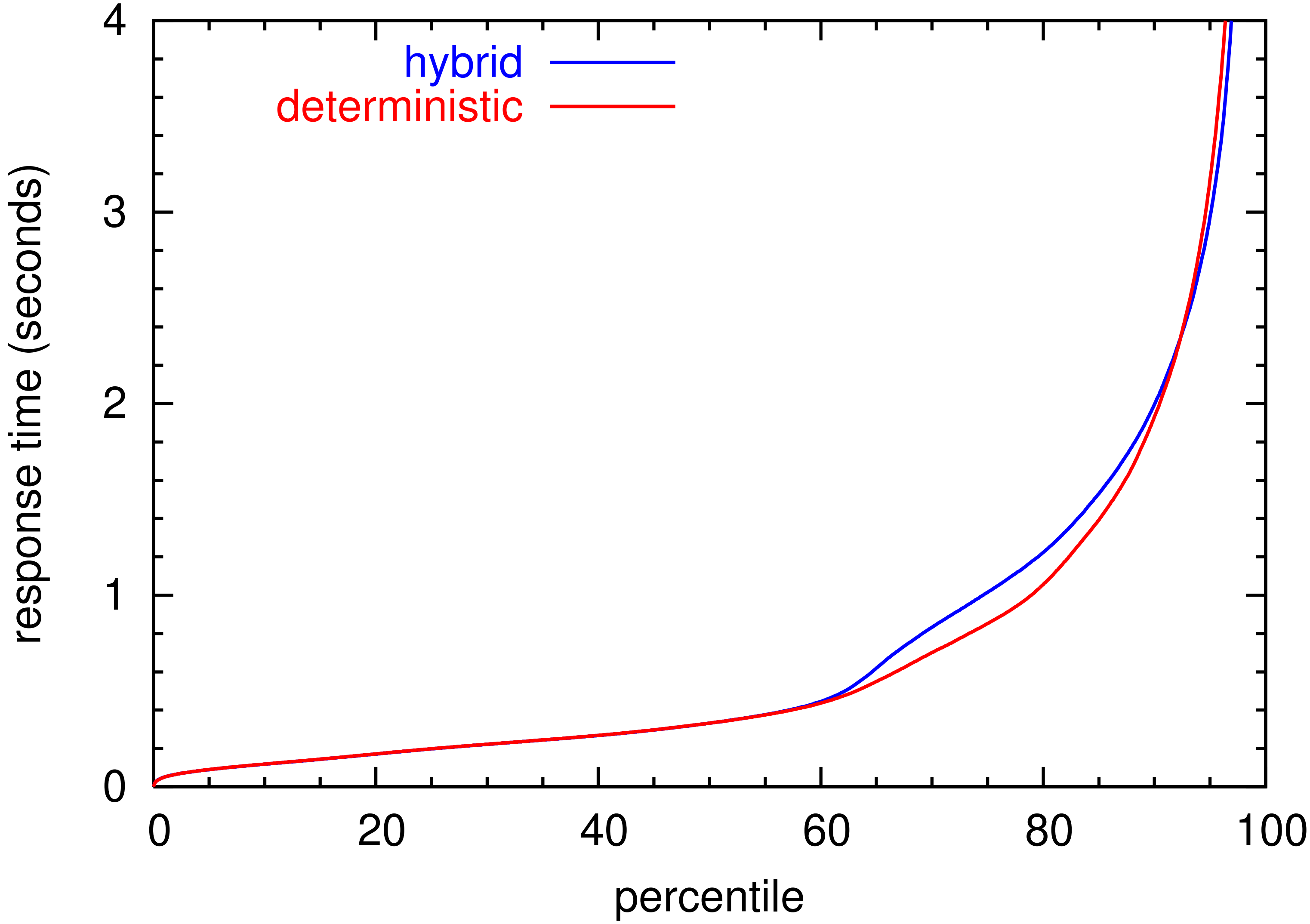
randomization reduces failure rate by about 1000x



Results (packet loss: lossy receivers, response time)

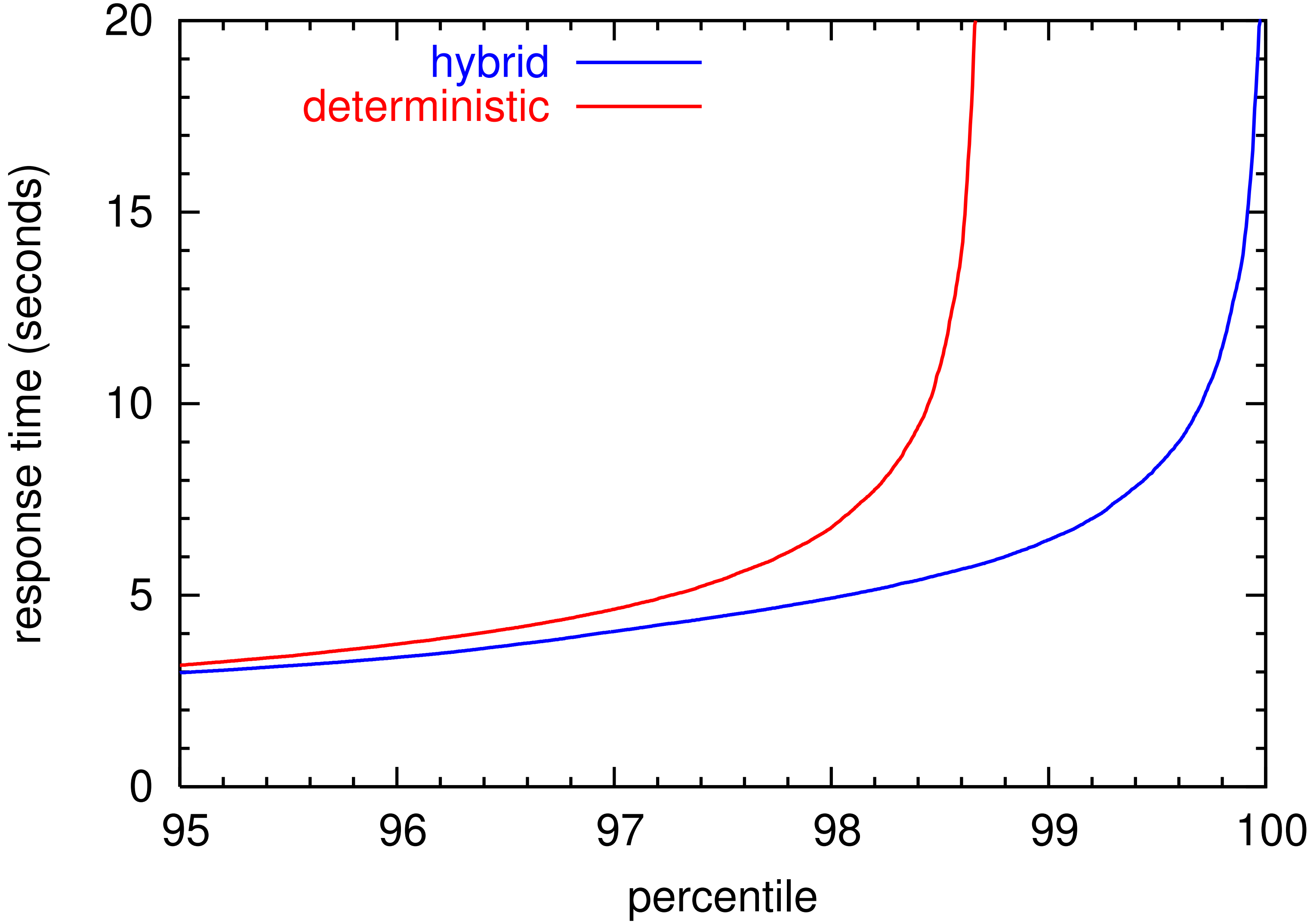
1/5 of nodes drop 1/2 of incoming packets

randomization incurs small penalty



Results (packet loss: lossy receivers, failure rate)

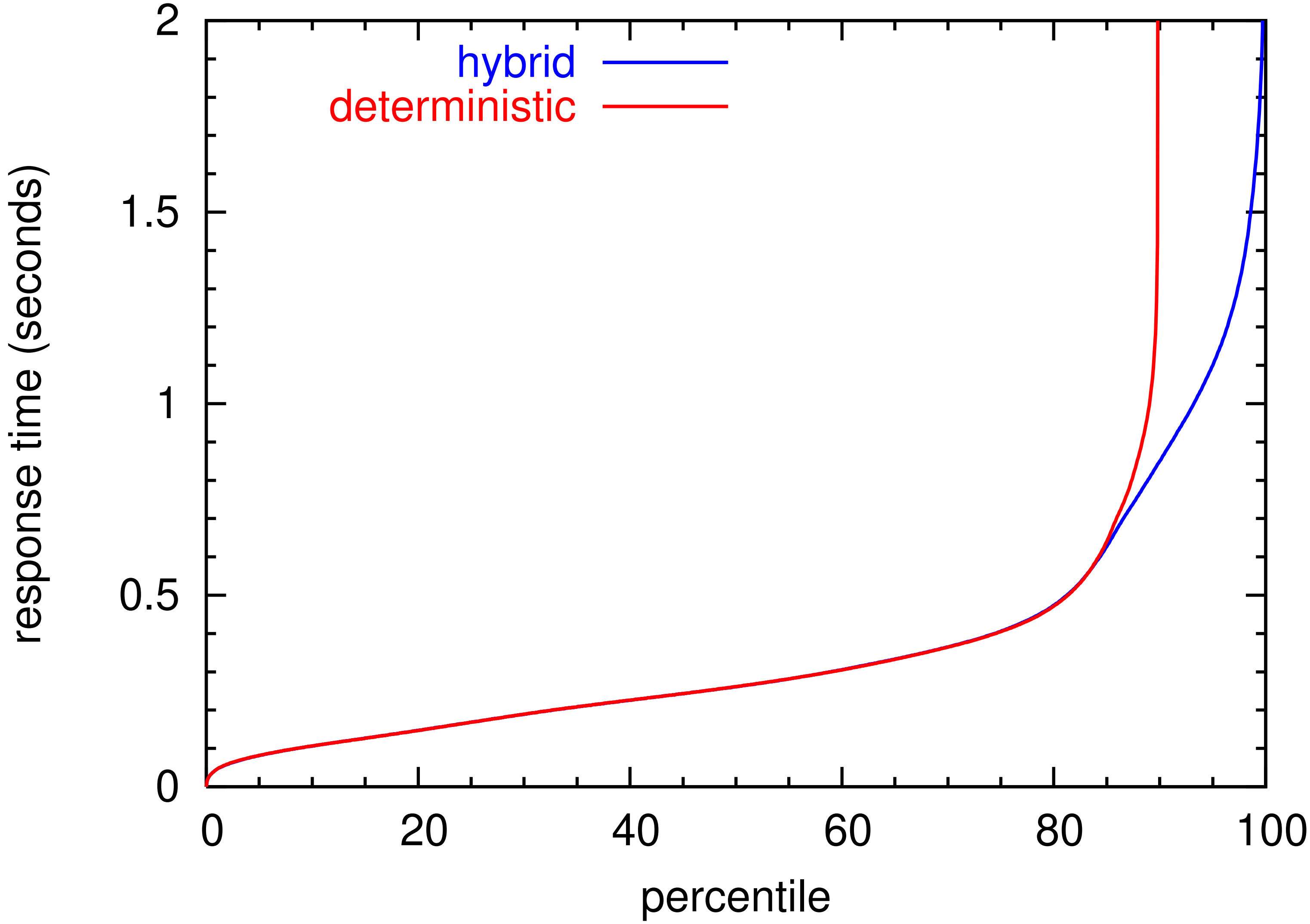
randomization reduces failure rate by more than 10x



Results (route loss: blind peers, response time)

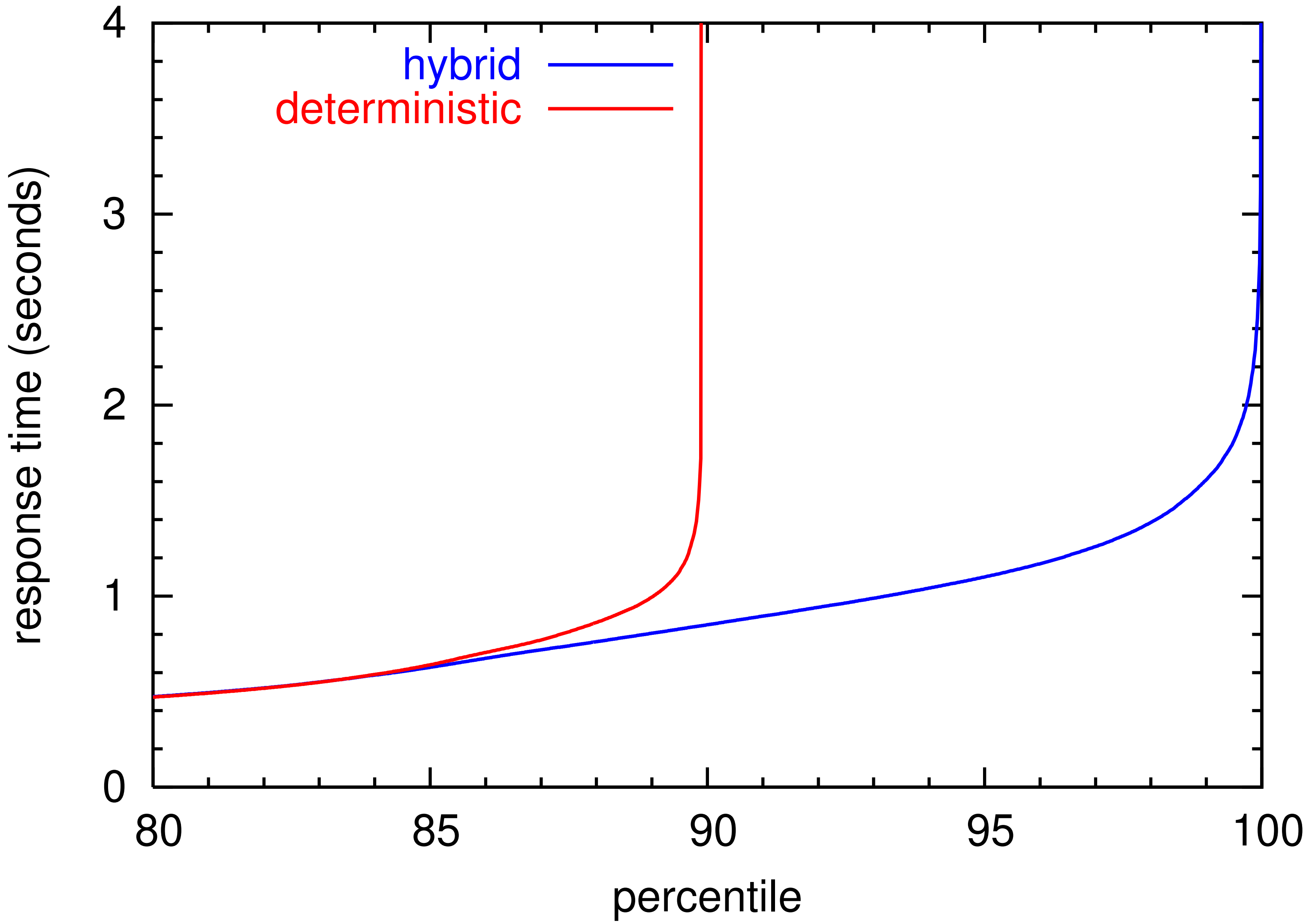
1/5 of nodes can reach only 1/20 of peers

no cost for hybrid randomization



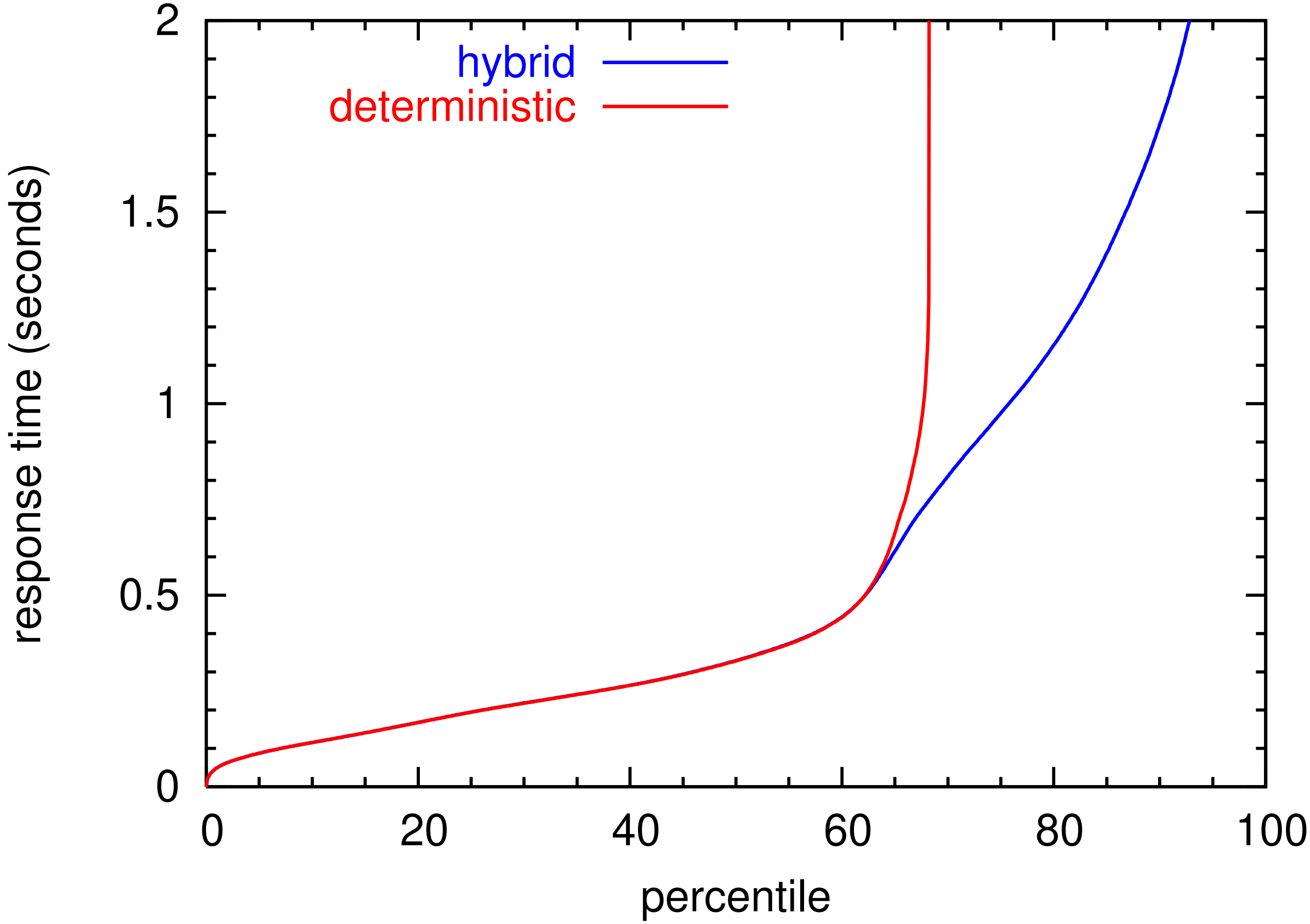
Results (route loss: blind peers, failure rate)

randomization reduces failure rate by more than 10,000x



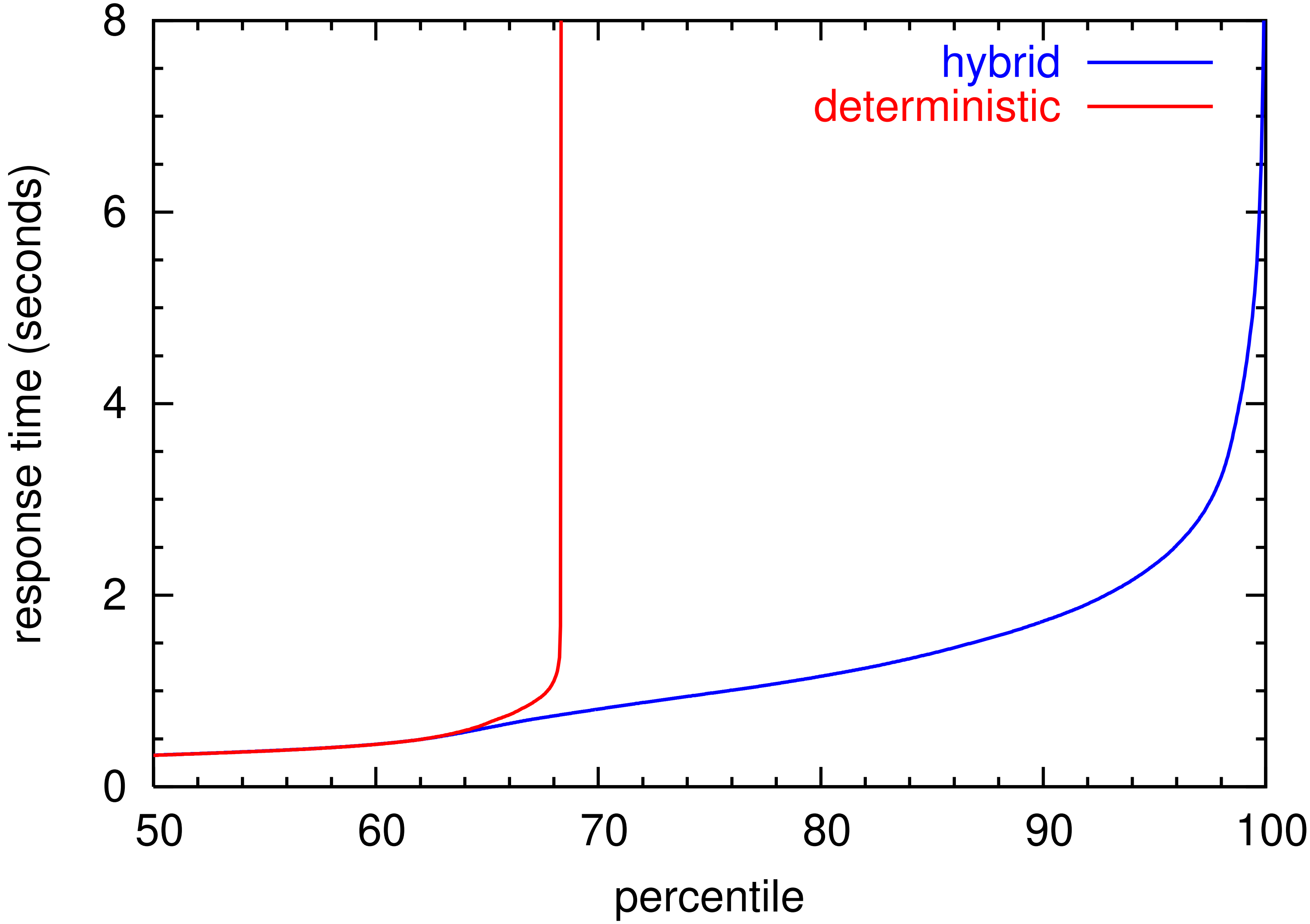
Results (bugs, response time)

1/10 of nodes buggy & unable to forward
no cost for hybrid randomization



Results (bugs, failure rate)

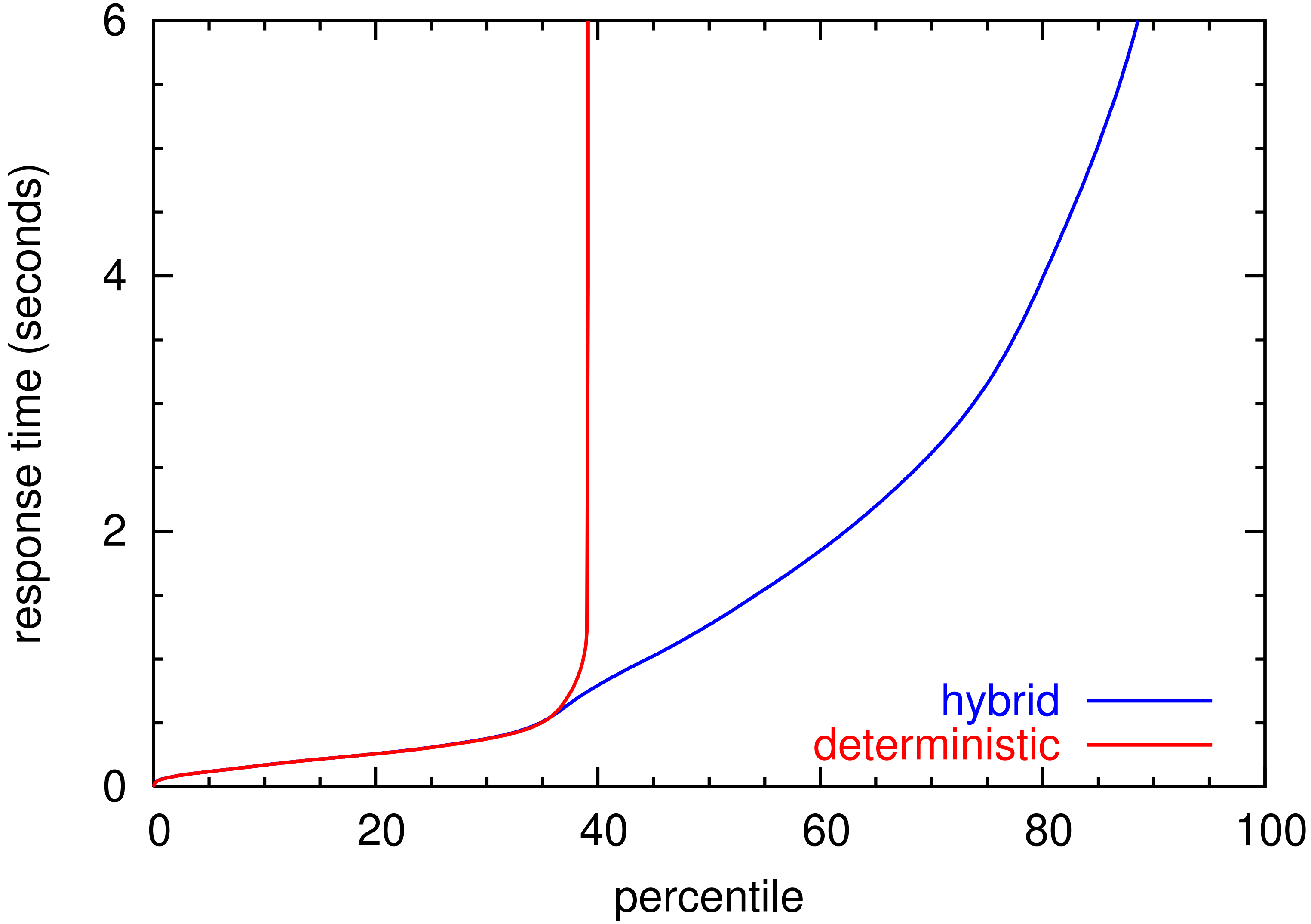
randomization reduces failure rate by more than 10,000x



Results (membership churn, response time)

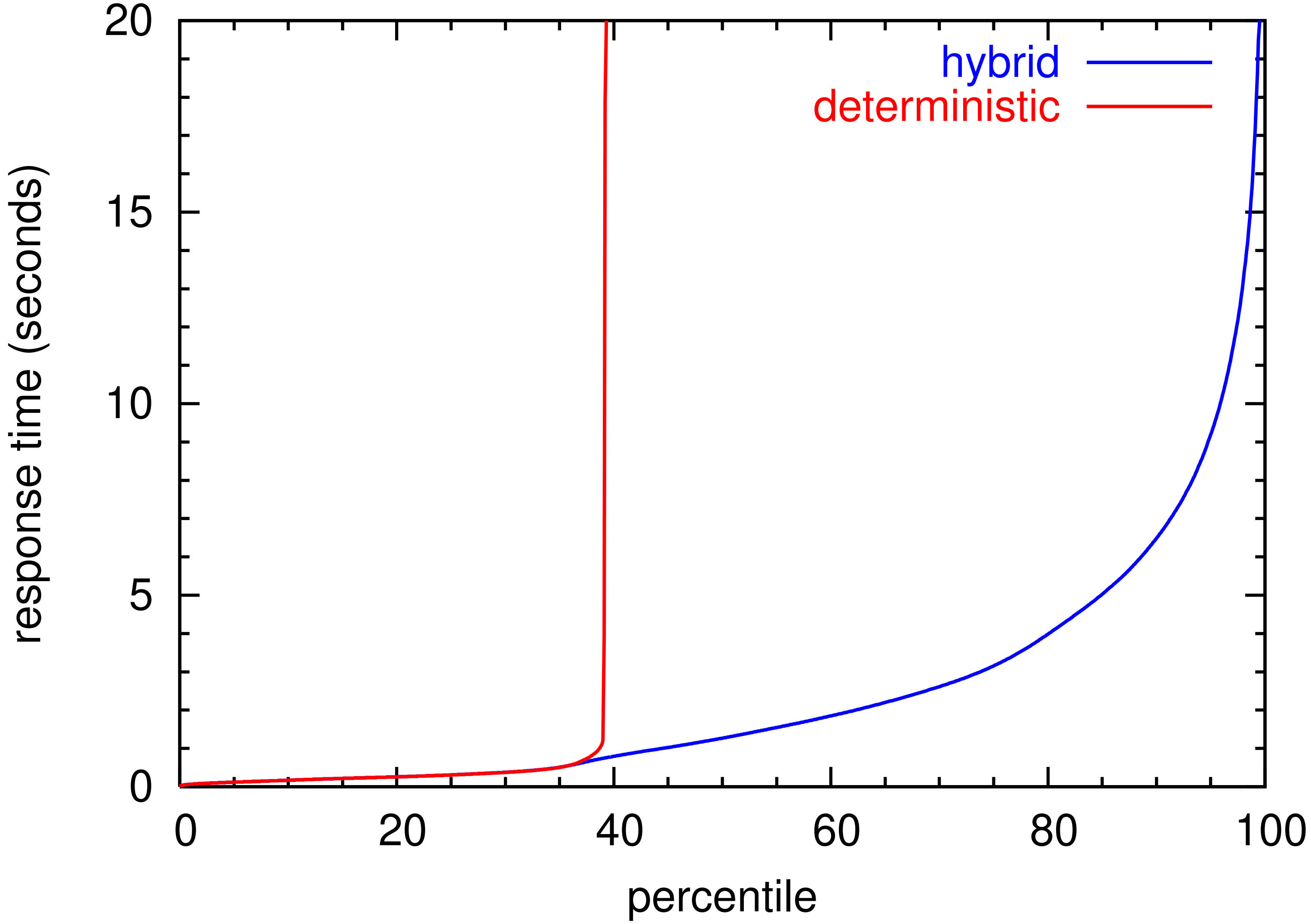
6/7 of nodes recently arrived, 1/2 of those already vanished

no cost for hybrid randomization



Results (membership churn, failure rate)

randomization reduces failure rate by more than 100x

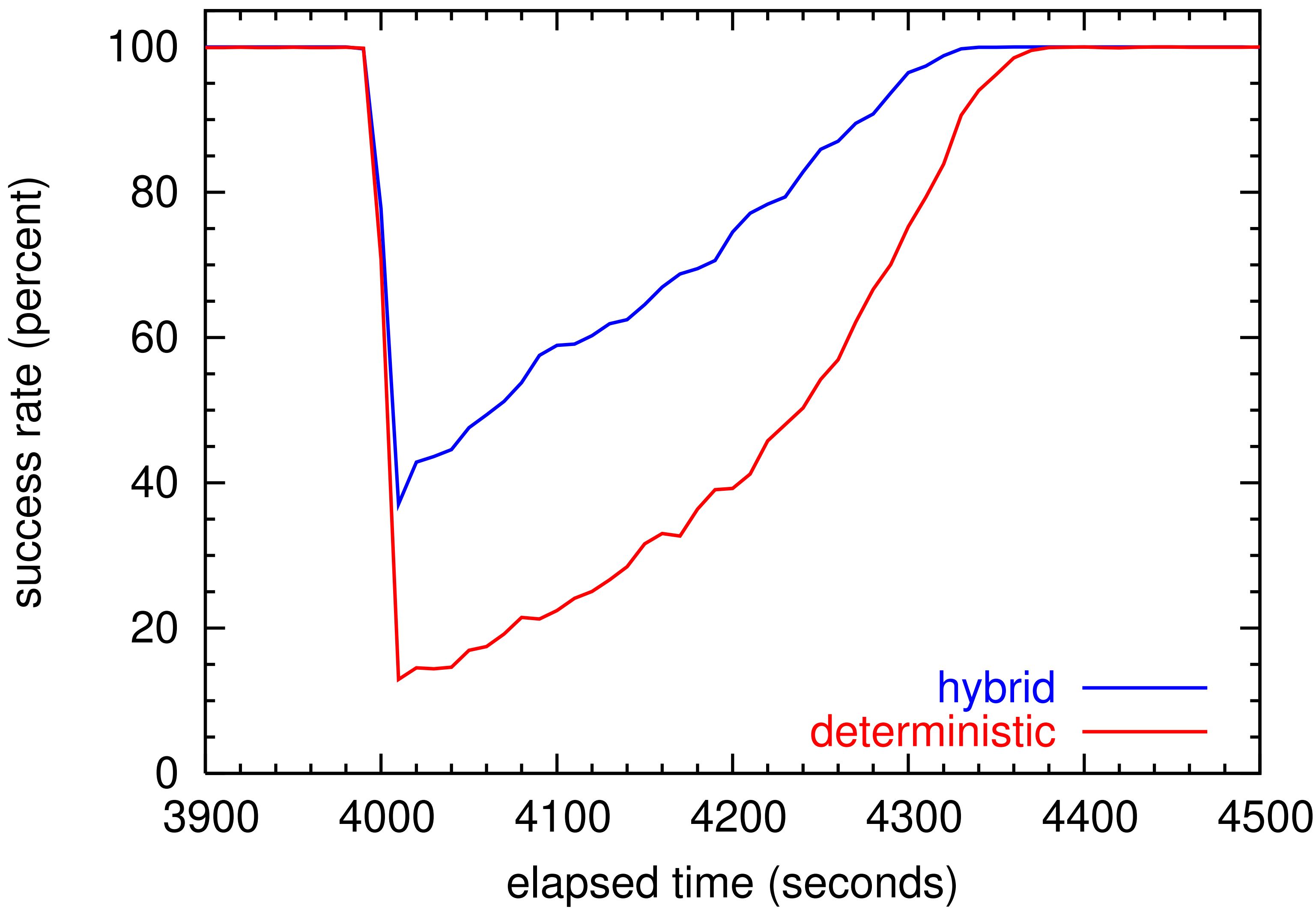


Experiments

steady-state faults	no faults
	packet loss (in network, at receivers, at senders)
	route loss (invisible peers, blind peers)
	bugs (peers that cannot forward)
	membership churn (transient peers)
dynamic faults	membership burst (mass join, mass leave)
scaling	settling time (time to construct routing tables)
	response time (mean, in equilibrium)

Results (mass leave)

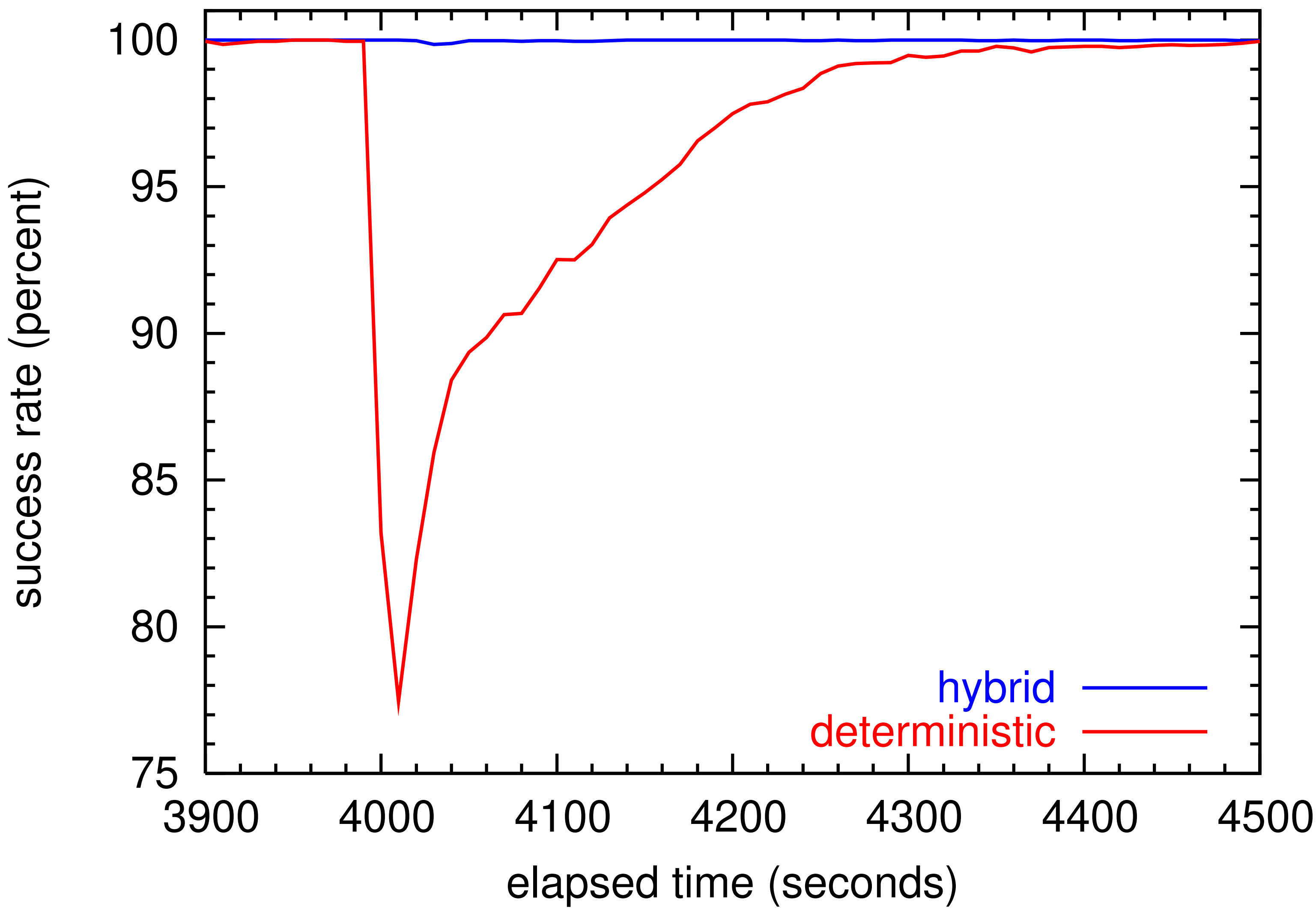
all nodes reach equilibrium, then 1/2 leave en masse
randomization has better success rate during recovery



Results (mass join)

1/2 of nodes reach equilibrium, then other 1/2 join en masse

randomization reduces initial failure rate by about 100x



DHT summary

DHT fundamental operation

send(label,message)

(find the node matching a label)

Tangle

randomizes forwarding & routing table construction

randomized forwarding experiments

greatly reduced failure rates

little or no performance penalty

for wide variety of undetected faults

Future work

Tangle parameters

real wide-area experiments

above Tangle: serverless web (Buzz)

distributed search index (much harder than DHT)

Conclusion

randomized forwarding:

  alternative to detecting & routing around faults

  preemptively route around everything

basis of new peer-to-peer protocols for very different applications

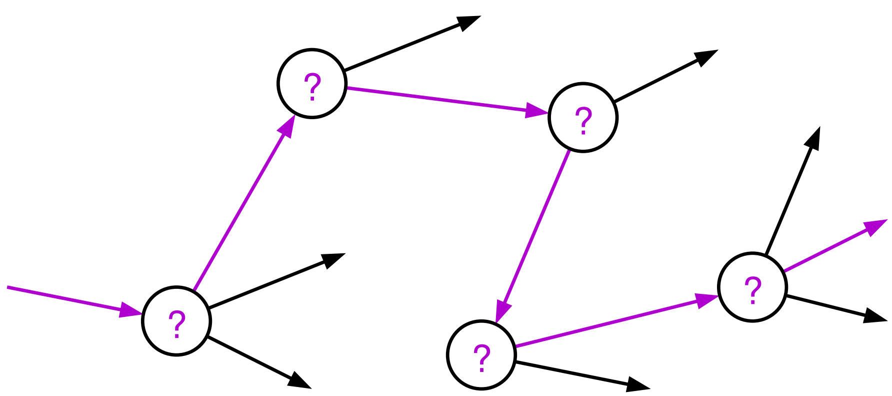
evaluated using analysis & simulation experiments

across protocols & across faults:

  high degree of fault tolerance

  little or no cost in performance

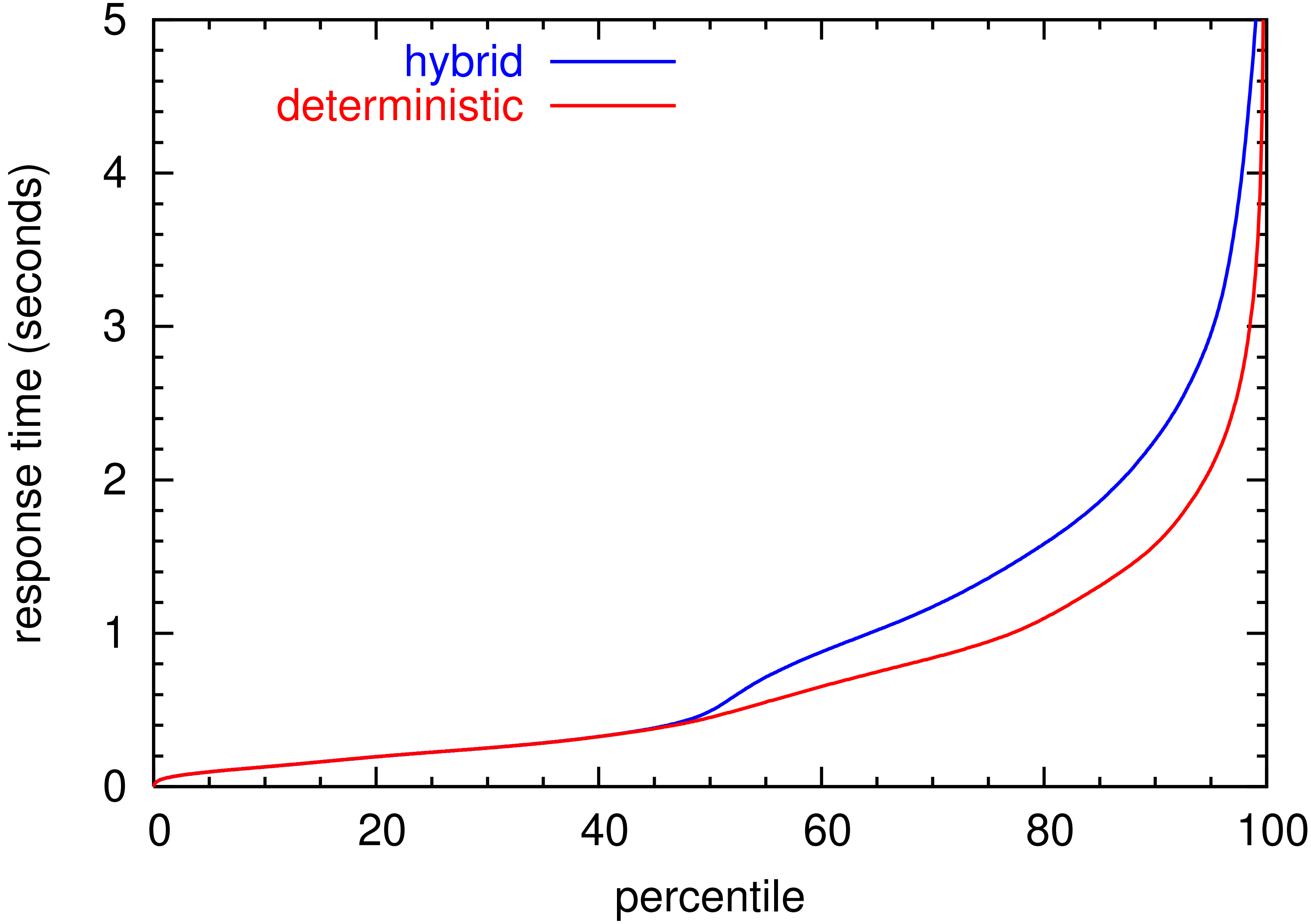
http://www.nicemice.net/amc/research/randomized-forwarding/



Results (packet loss: lossy network, response time)

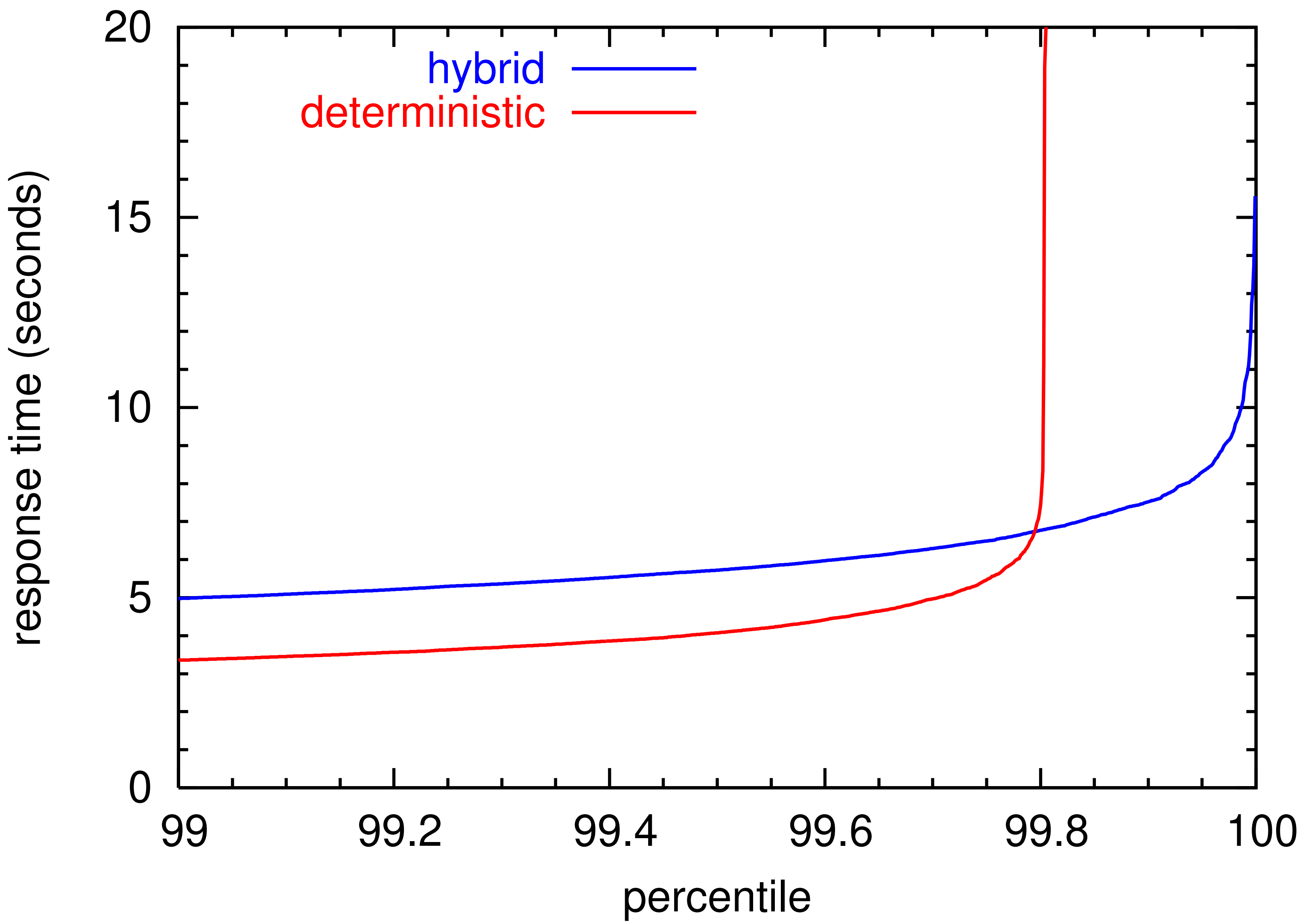
1/10 of all packets dropped

randomization incurs small penalty



Results (packet loss: lossy network, failure rate)

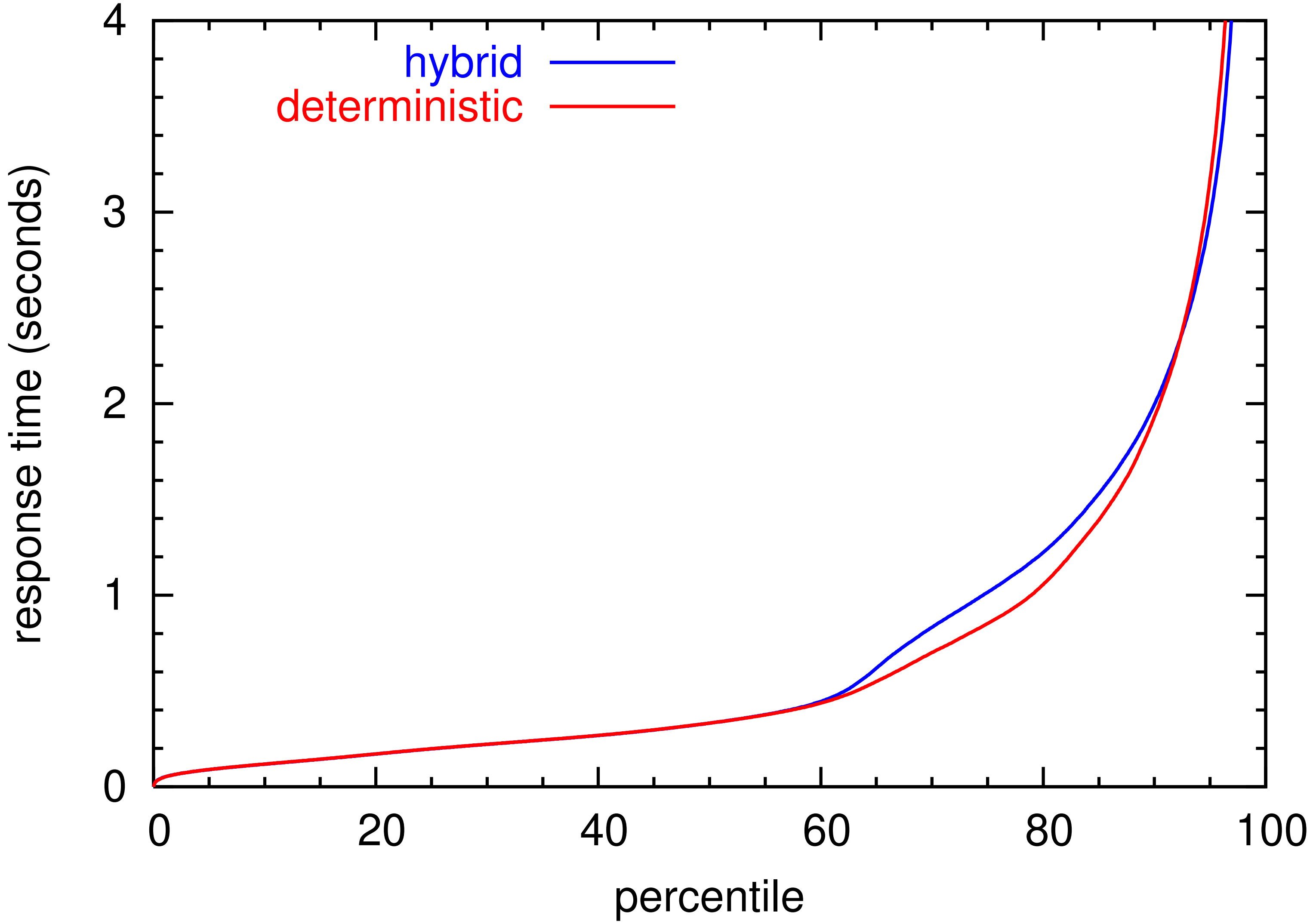
randomization reduces failure rate by more than 100x



Results (packet loss: lossy senders, response time)

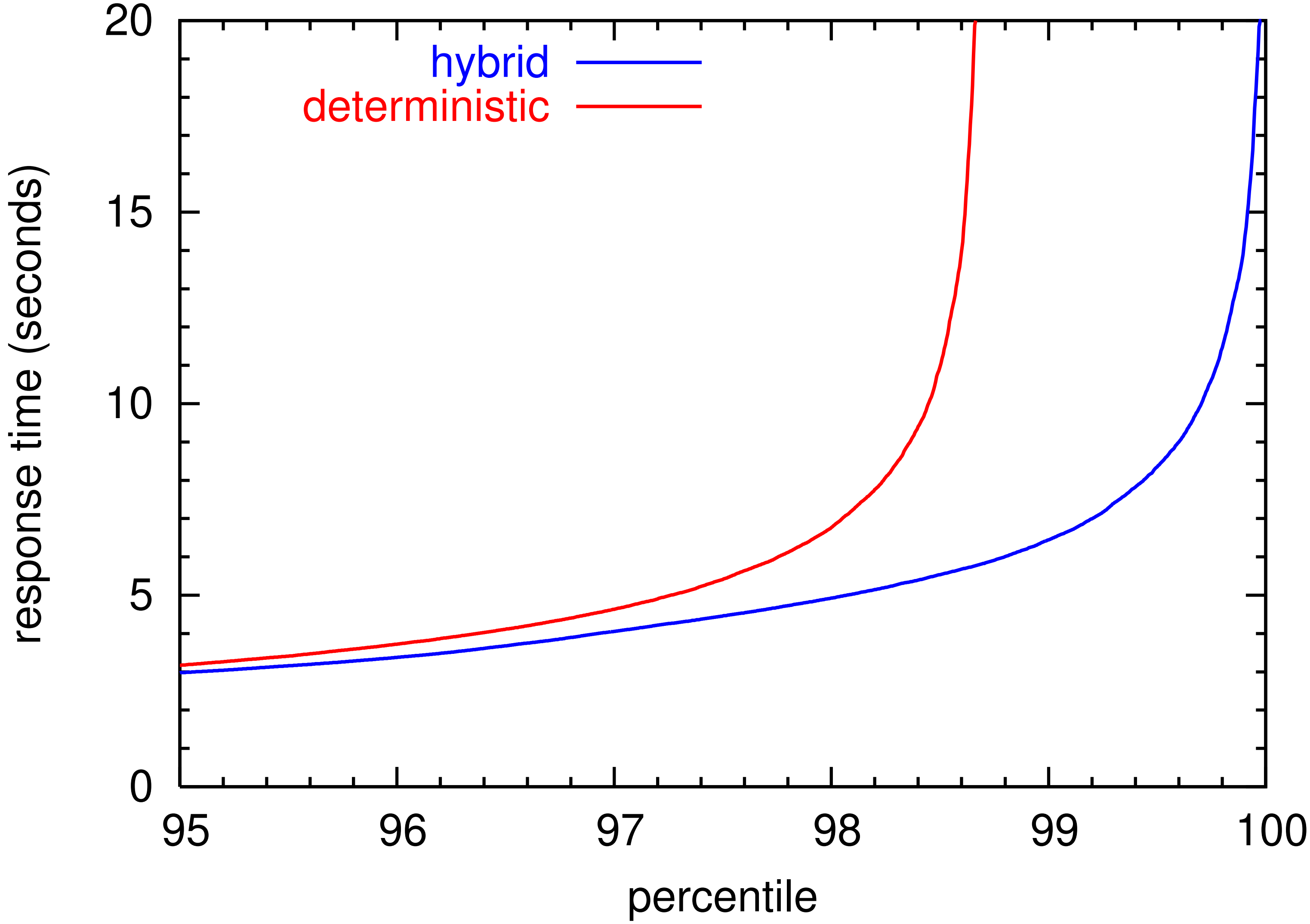
1/5 of nodes drop 1/2 of outgoing packets

randomization incurs small penalty



Results (packet loss: lossy senders, failure rate)

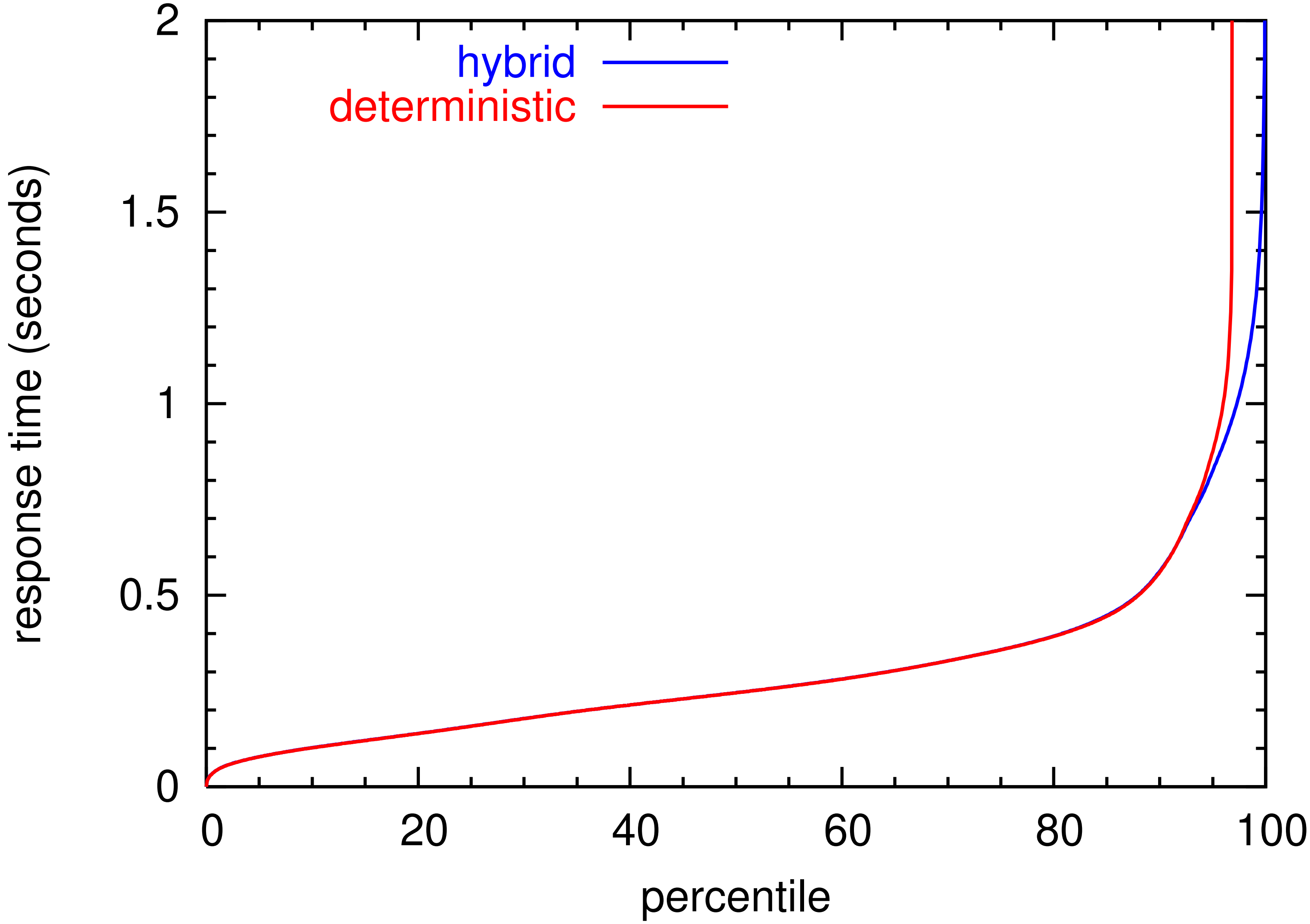
randomization reduces failure rate by more than 10x



Results (route loss: invisible peers, response time)

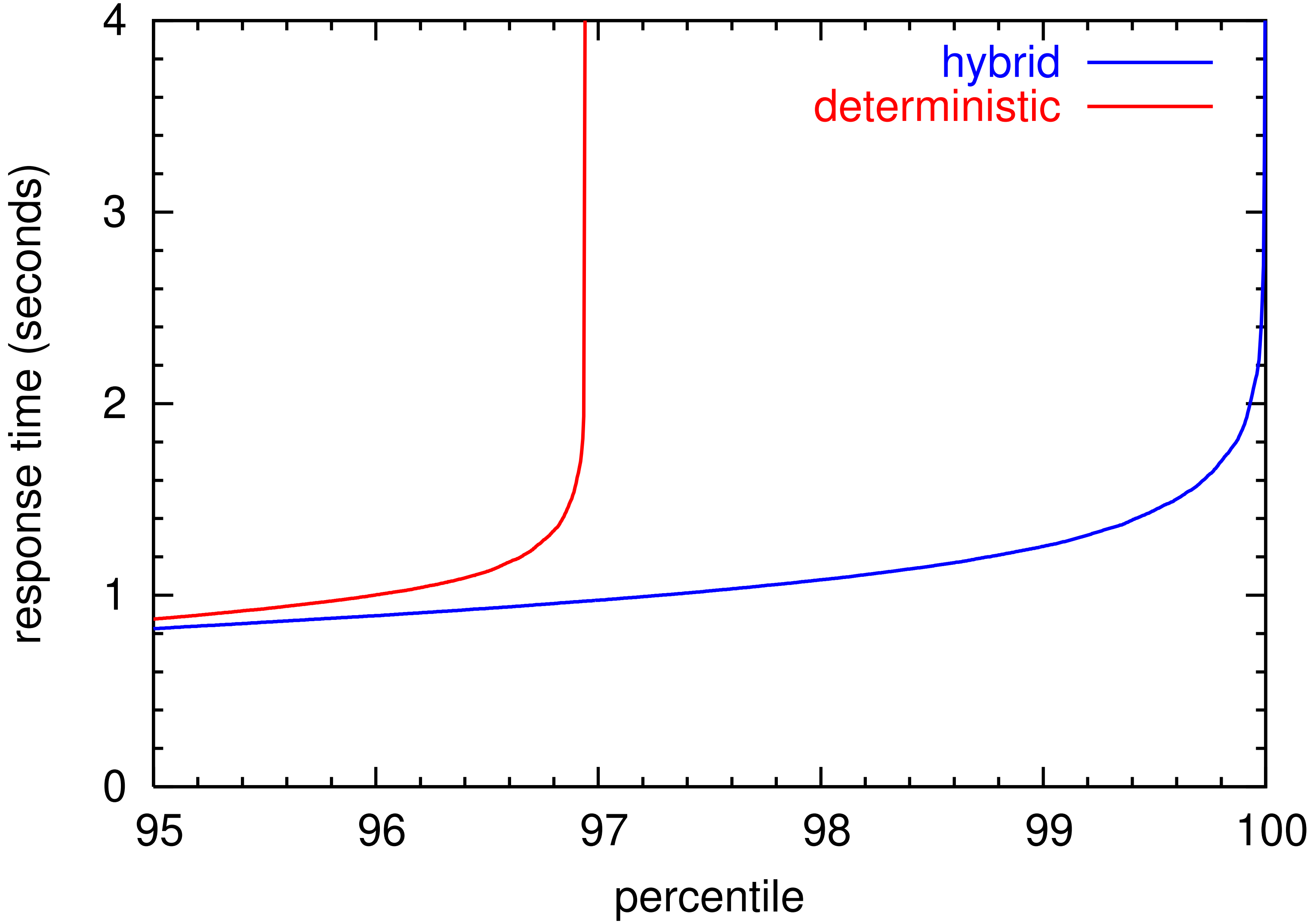
1/5 of nodes reachable by only 1/20 of peers

no cost for hybrid randomization



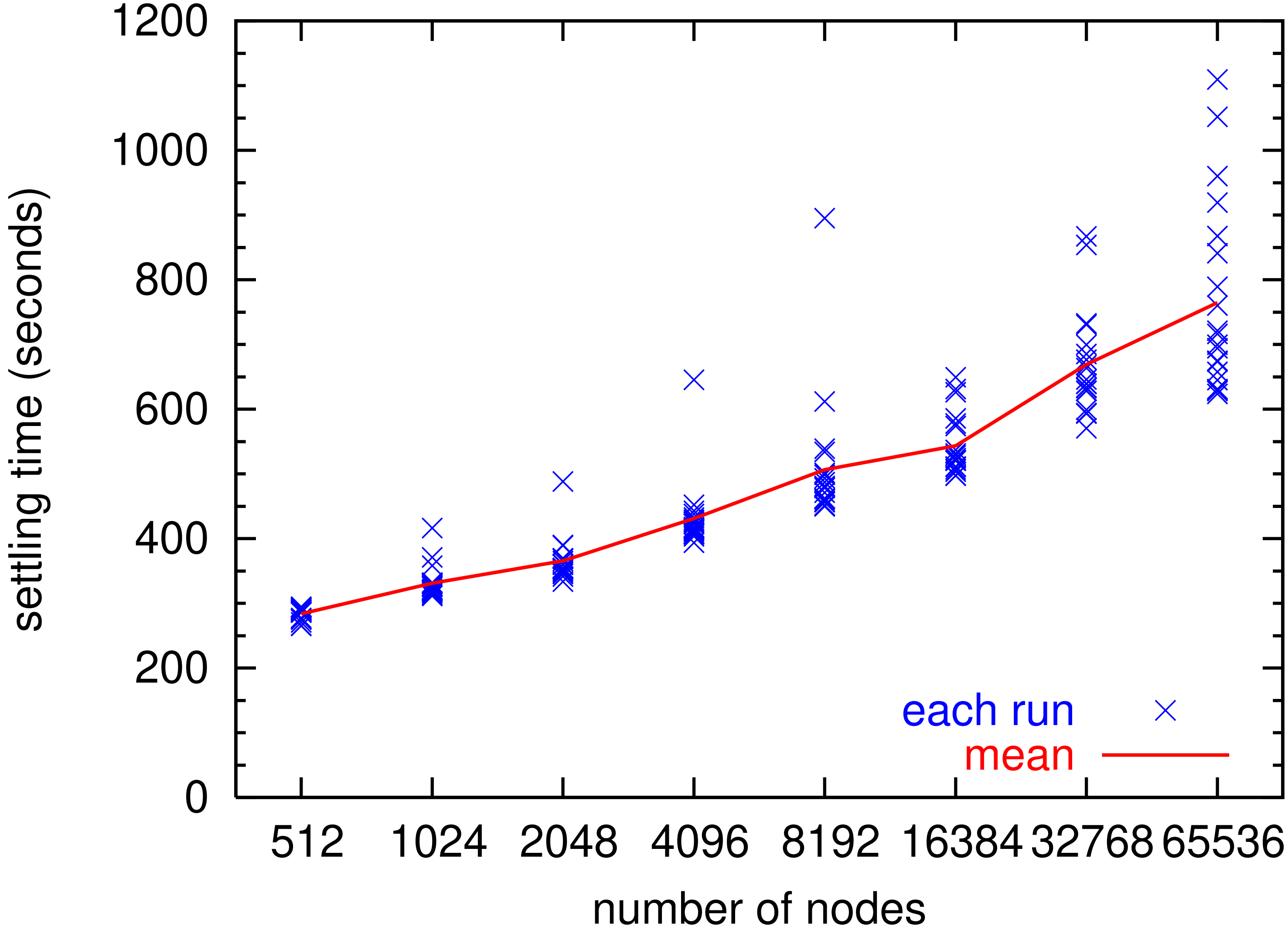
Results (route loss: invisible peers, failure rate)

randomization reduces failure rate by more than 10,000x



Results (scaling: settling time)

time to build all routing tables from scratch



Results (scaling: mean response time)