# Minimizing Churn in Distributed Systems

Brighten Godfrey
Scott Shenker
Ion Stoica

SIGCOMM 2006

# introduction

- Churn: an important factor for most distributed systems
  - Turnover causes dropped requests, increased bandwidth, ...
- Would like to optimize for stability
  - Select which nodes to use

Can't prevent a node from failing, but we can select which nodes to use

# introduction

- Past work uses heuristics for specific systems
- Our goal: a **general** study of minimizing churn
  - How can we select nodes to minimize churn?
  - Can we characterize how existing systems select nodes and the impact on their performance?

...applicable to a wide range of systems

# contents

- **an example system**
- evaluation of node selection strategies

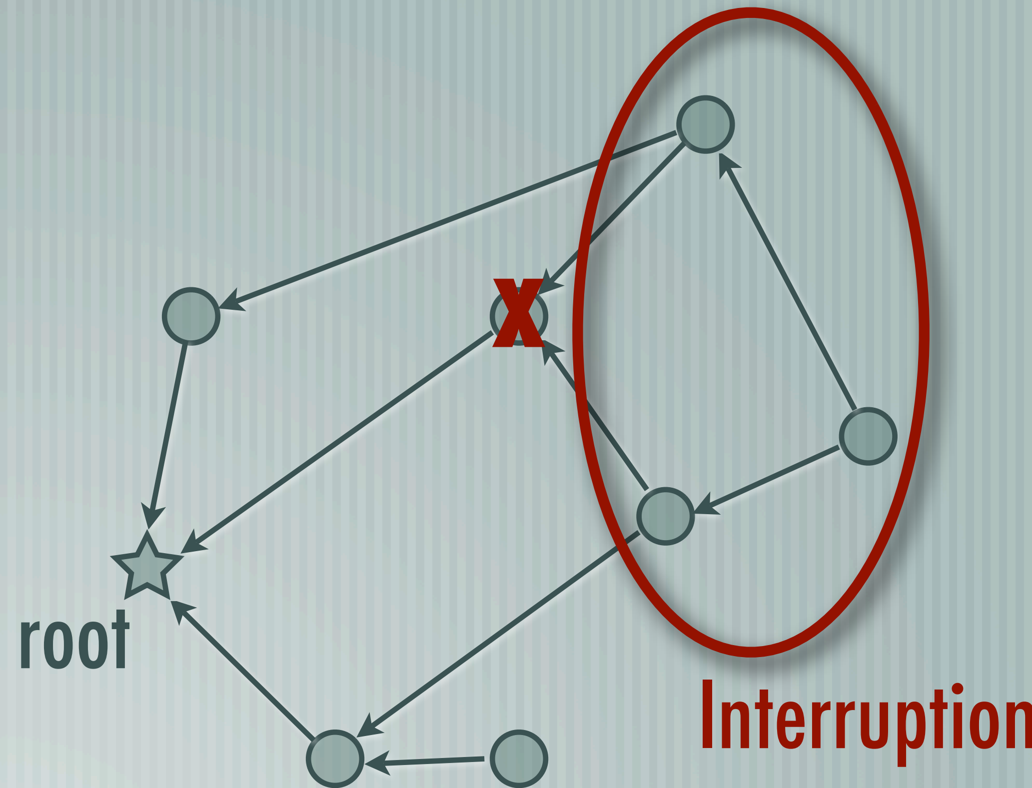
  *(how can we minimize churn?)*
- applications

  *(how do existing systems select nodes?)*
- conclusions

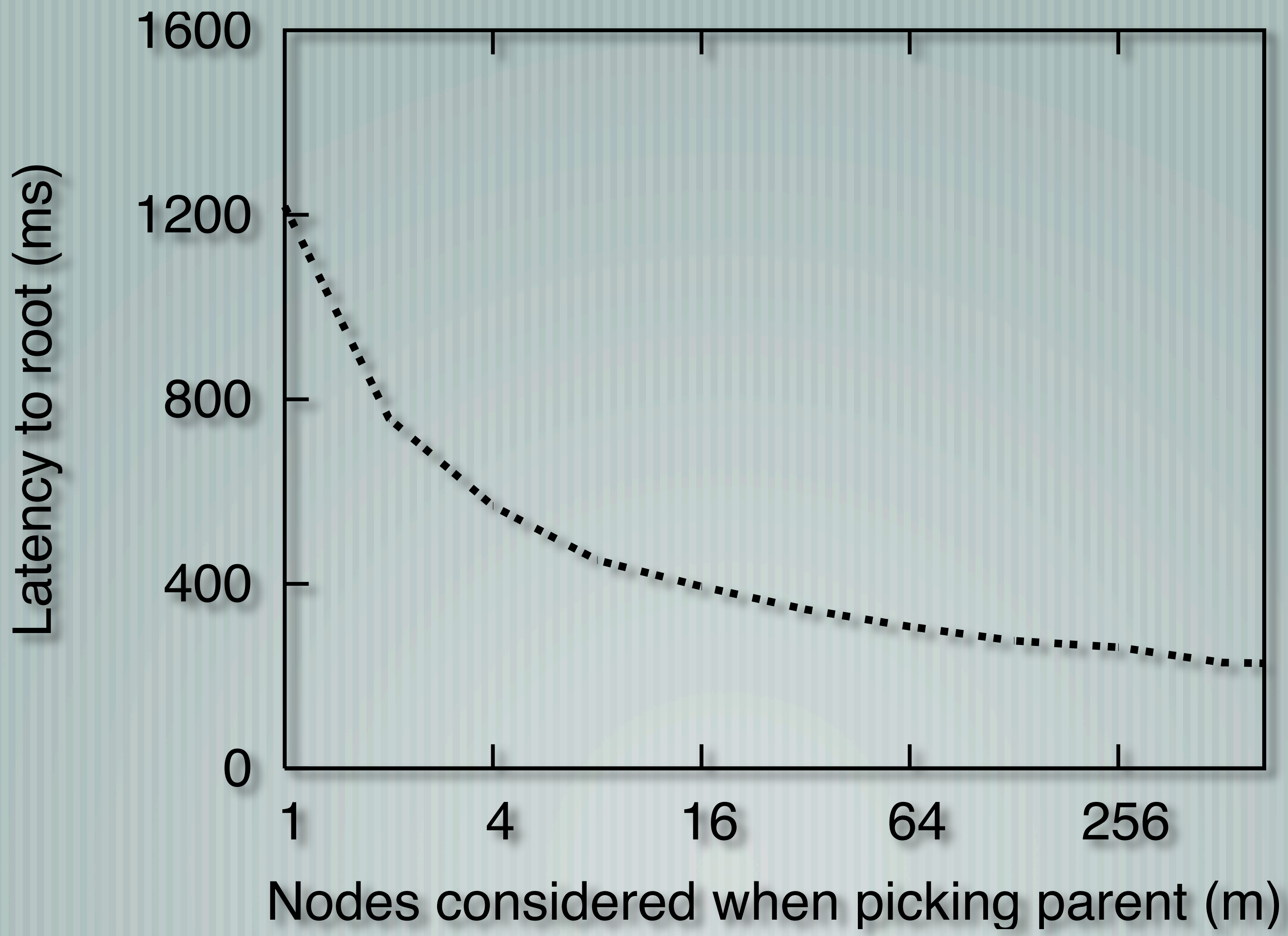
# example: overlay multicast

Join:

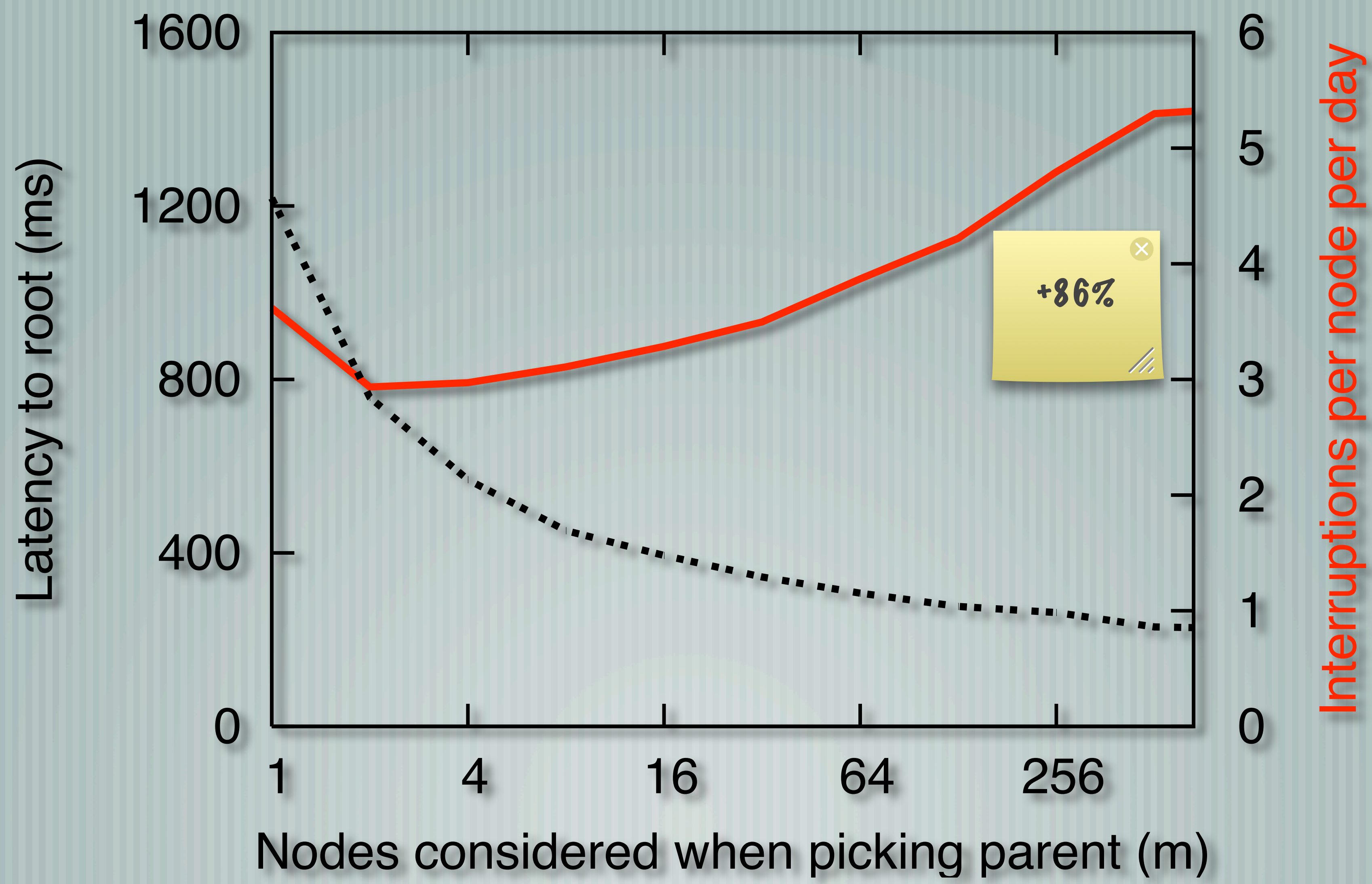
- Consider *m* random nodes with # children < max
- Pick one as parent to minimize latency to root



# example: overlay multicast



# example: overlay multicast



# example: overlay multicast

In terms of interruption rate,

**Random Replacement** of parent (m=1)

better than

**Preference List** selection (large m)

Why?

# contents

- an example system
- **evaluation of node selection strategies**

  *(how can we minimize churn?)*
- applications

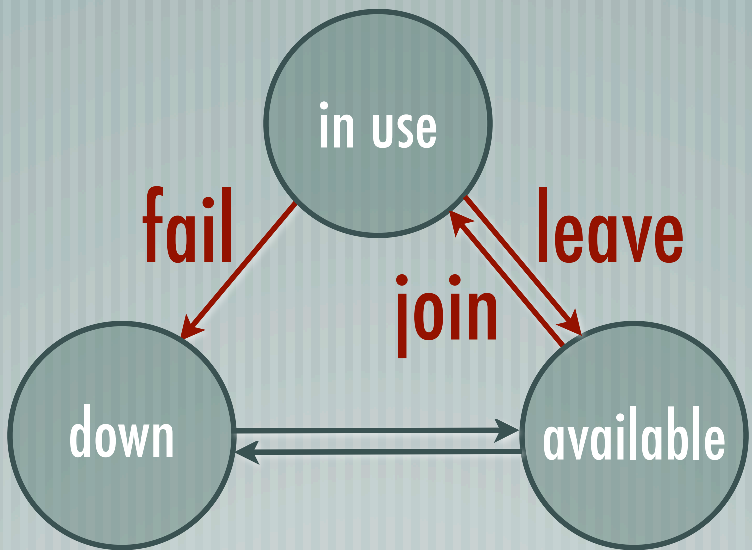
  *(how do existing systems select nodes?)*
- conclusions

# the core problem

- Node selection task
  - *n* nodes available
  - pick *k* to be "in use"
  - when one fails, pick a replacement
- Minimize **churn**: rate of change in set of in-use nodes

# defining churn

- For each node:




churn += $\frac{1}{k}$

$k$ = # of nodes in use

- Intuition: when a node joins or leaves a DHT, $1/k$ of stored objects change ownership
- ...then divide by runtime

# node selection strategies

Predictive

- Longest uptime
- Most available
- Max expectation
- ...

Agnostic

- Random Replacement
- Preference List

# agnostic selection strategies

Random Replacement

Select random available node to replace failed node

Passive Preference List

Rank nodes (e.g. by latency); Select most preferred as replacement

Active Prefe

Pref List is:
(1) essentially static across time
(2) essentially unrelated to churn

...and switch to more preferred nodes when they join

# evaluation

churn

Active PL

Passive PL

$2.5\text{-}8\times$

$1.2\text{-}3\times$

Random Replacement

$1.2\text{-}2.2\times$

Longest Uptime, Max Expectation

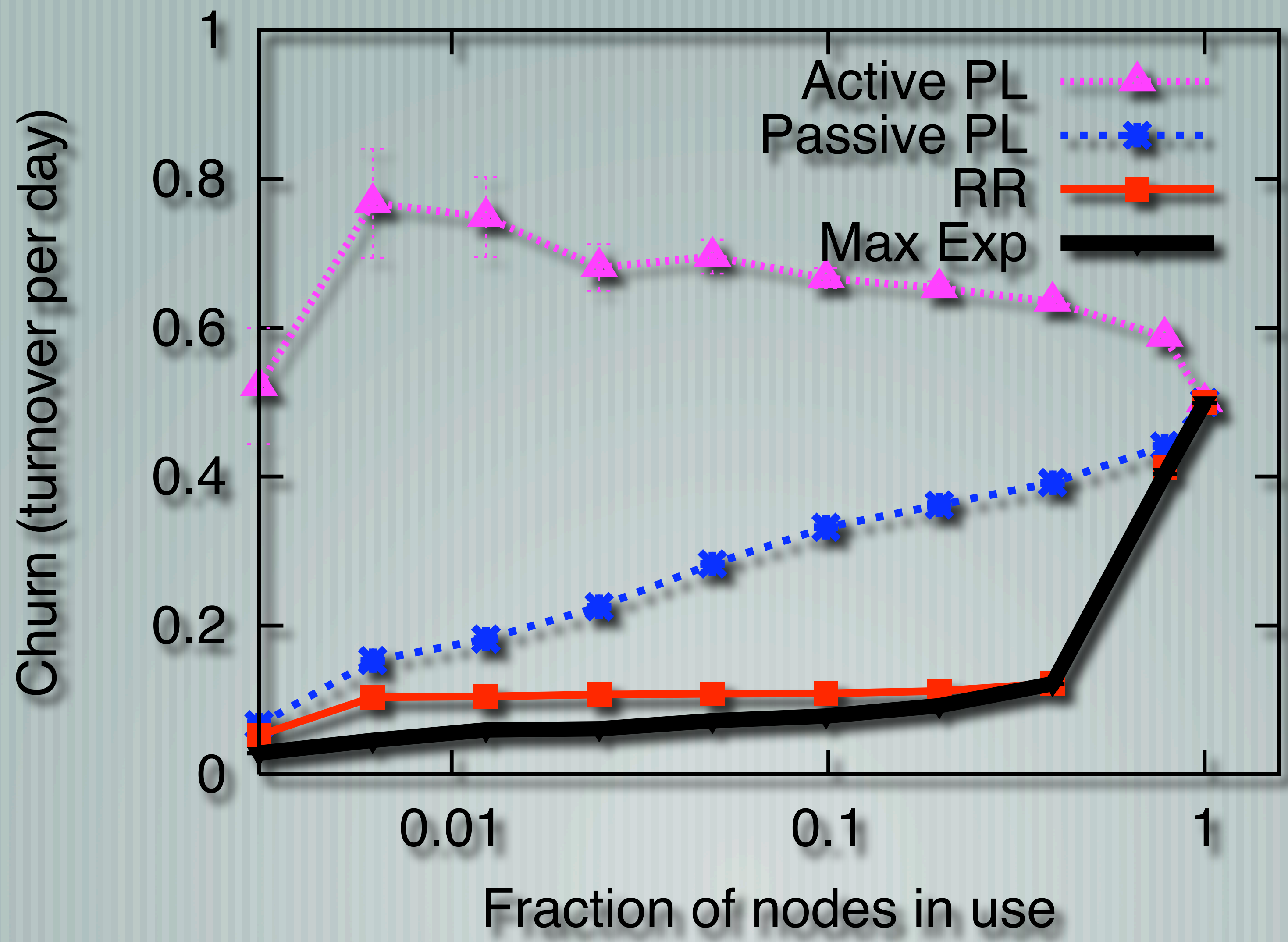
Why such a difference?

...even though neither uses history?

# evaluation

- 5 traces of node availability
  - PlanetLab [Stribling 2004-05]
  - Web sites [Bakkaloglu et al 2002]
  - Microsoft PCs [Bolosky et al 2000]
  - Skype superpeers [Guha et al 2006]
  - Gnutella peers [Saroiu et al 2002]
- Main conclusions held in all cases

# evaluation: PlanetLab trace



# intuition: PL

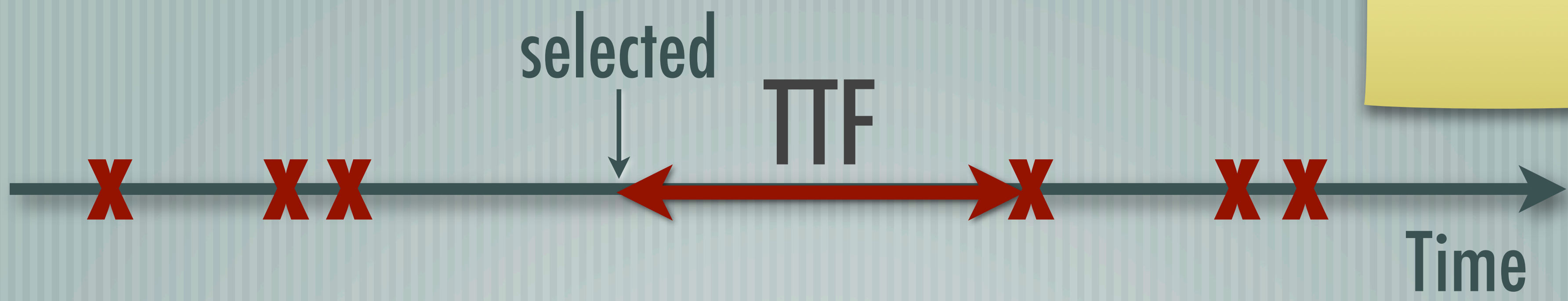
- uses the top *k* nodes in the preference list
- preference list unrelated to stability
- failure rate is about mean node failure rate

<--- becomes more and more true for Passive as k increases

# intuition: RR

An example of the classic "inspection paradox"

- RR like picking a node at a random time




session = time between 2 failures

- Long sessions occupy more time (trivially)
- So, RR biased towards landing in longer sessions
- Failure rate can be arbitrarily lower than mean

but it depends on the session time distribution

# RR vs. PL: analysis

$$E[C] = \frac{2}{\alpha d}\sum_{i=1}^{d}\frac{1}{\mu_i}\left(1 - \mathrm{E}\left[\exp\left\{-\frac{\alpha}{2(1-\alpha)}E[C]\cdot L_i\right\}\right]\right)$$

- Churn of RR decreases as session time distributions become "more skewed" (=> higher variance)

- RR can never have more than 2x the churn of PL strategies

# contents

- an example system
- evaluation of node selection strategies

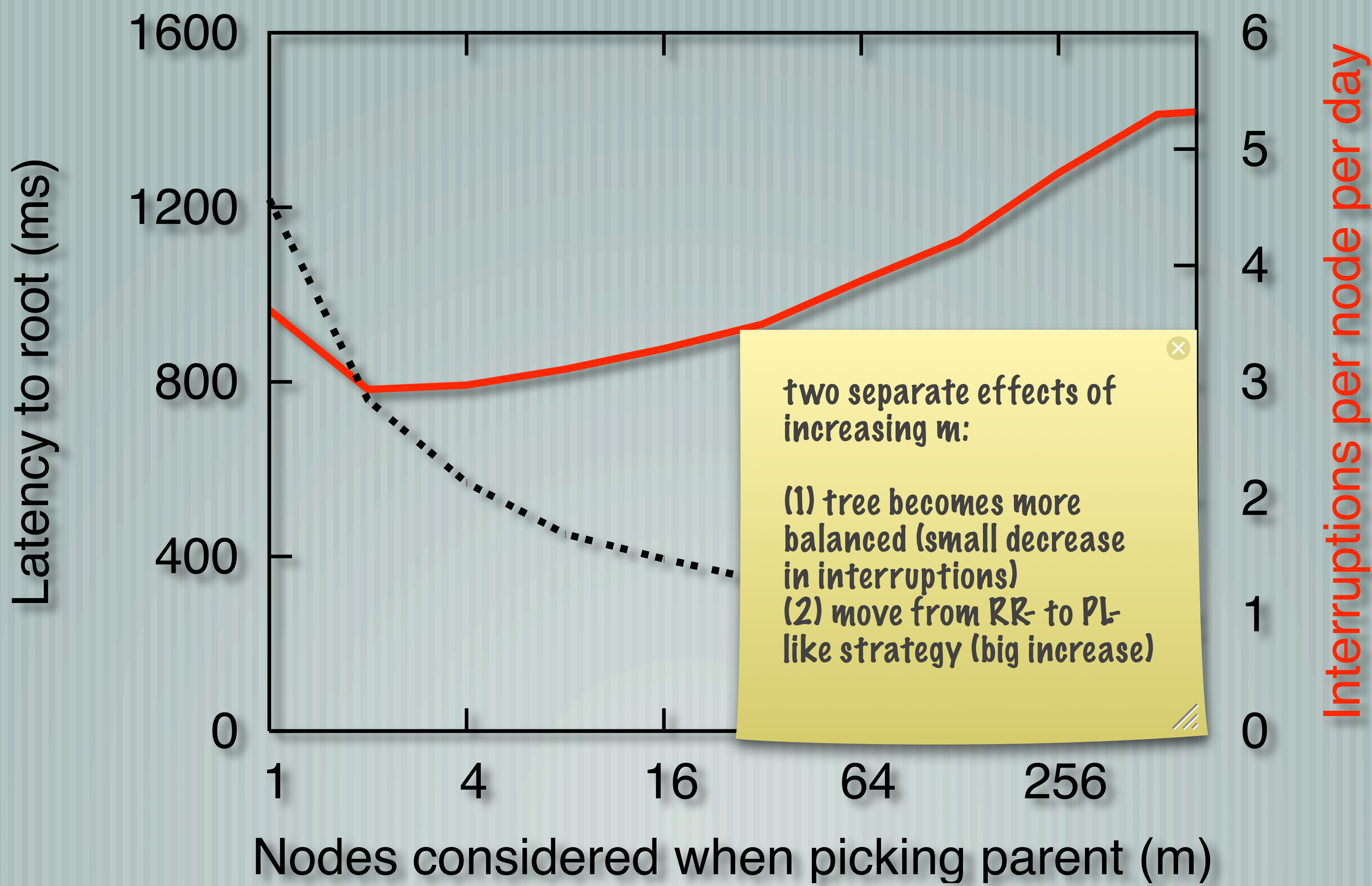
  *(how can we minimize churn?)*
- **applications**

  *(how do existing systems select nodes?)*
- conclusions

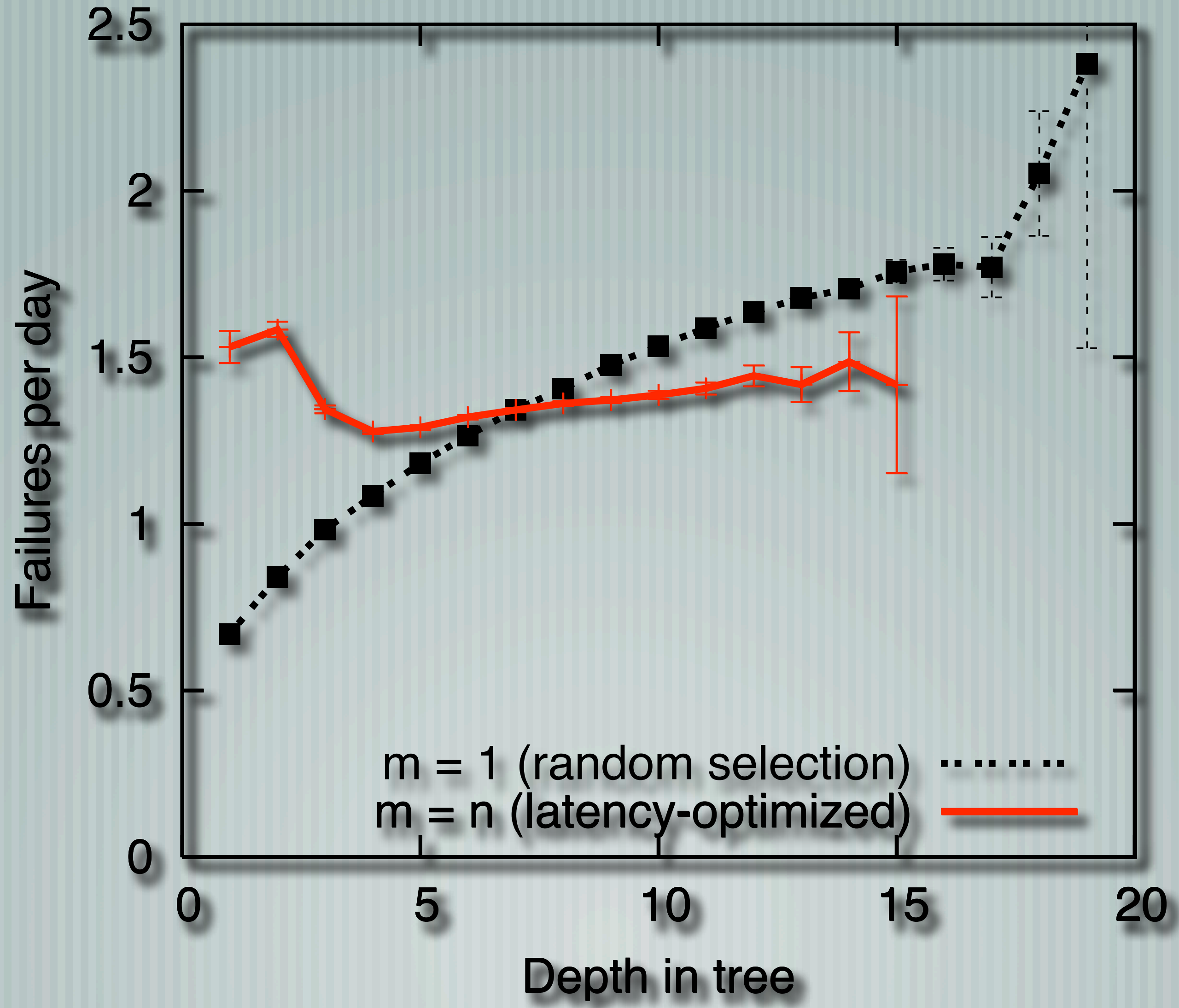
# applications of RR & PL

- anycast
- DHT replica placement
- overlay multicast
- DHT neighbor selection

# overlay multicast



# a peek inside the tree



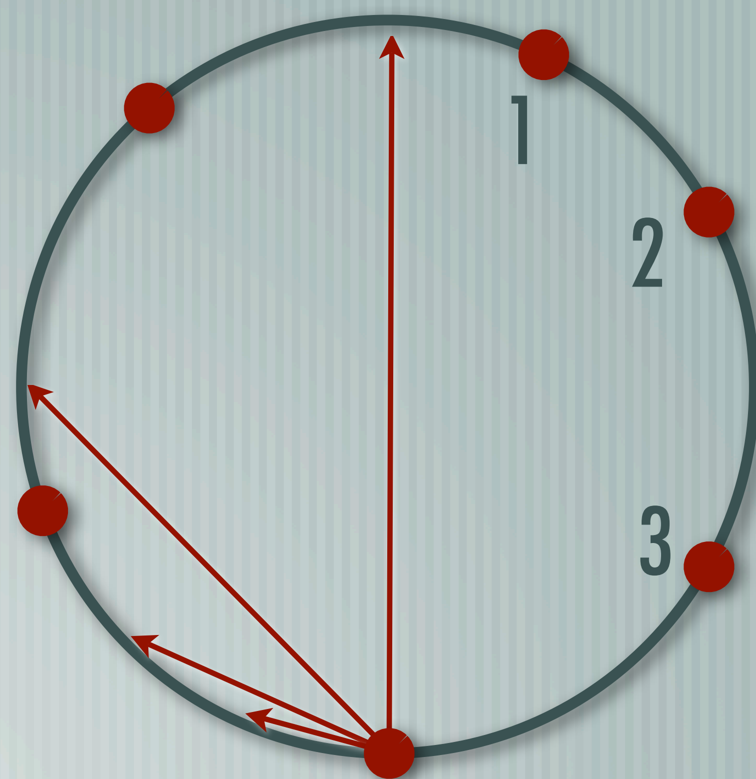
# overlay multicast notes

- Basic framework from [Sripanidkulchai et al SIGCOMM'04]
- Found random parent selection surprisingly good
- Tested 2 other heuristics to minimize interruptions
  - Both can perform better with some randomization!

# DHT neighbor selection

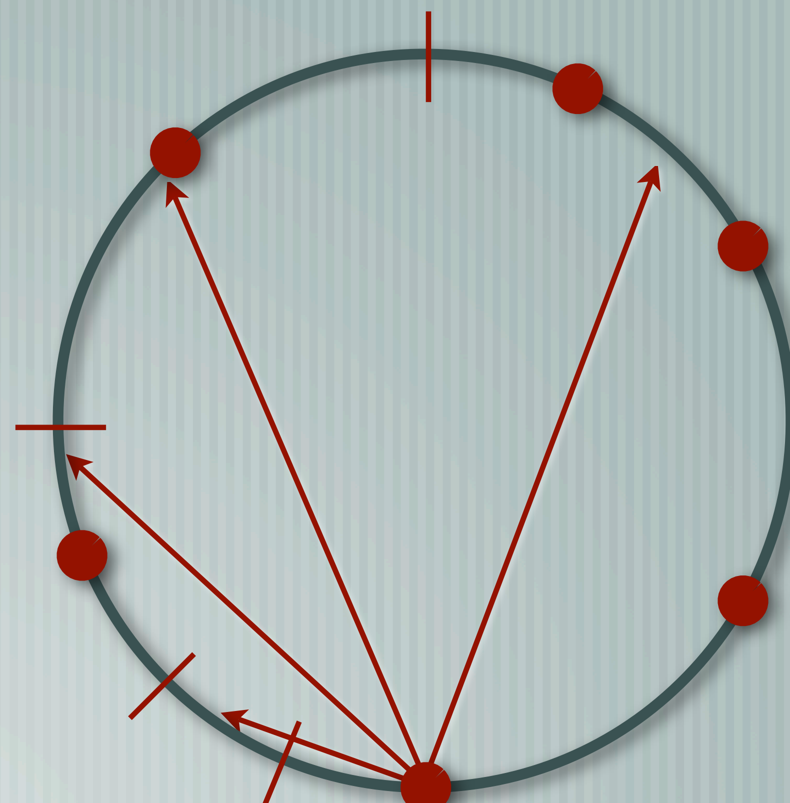
- Active PL strategy for selecting each finger
- Preference List arises accidentally

Standard Chord topology



# DHT neighbor selection

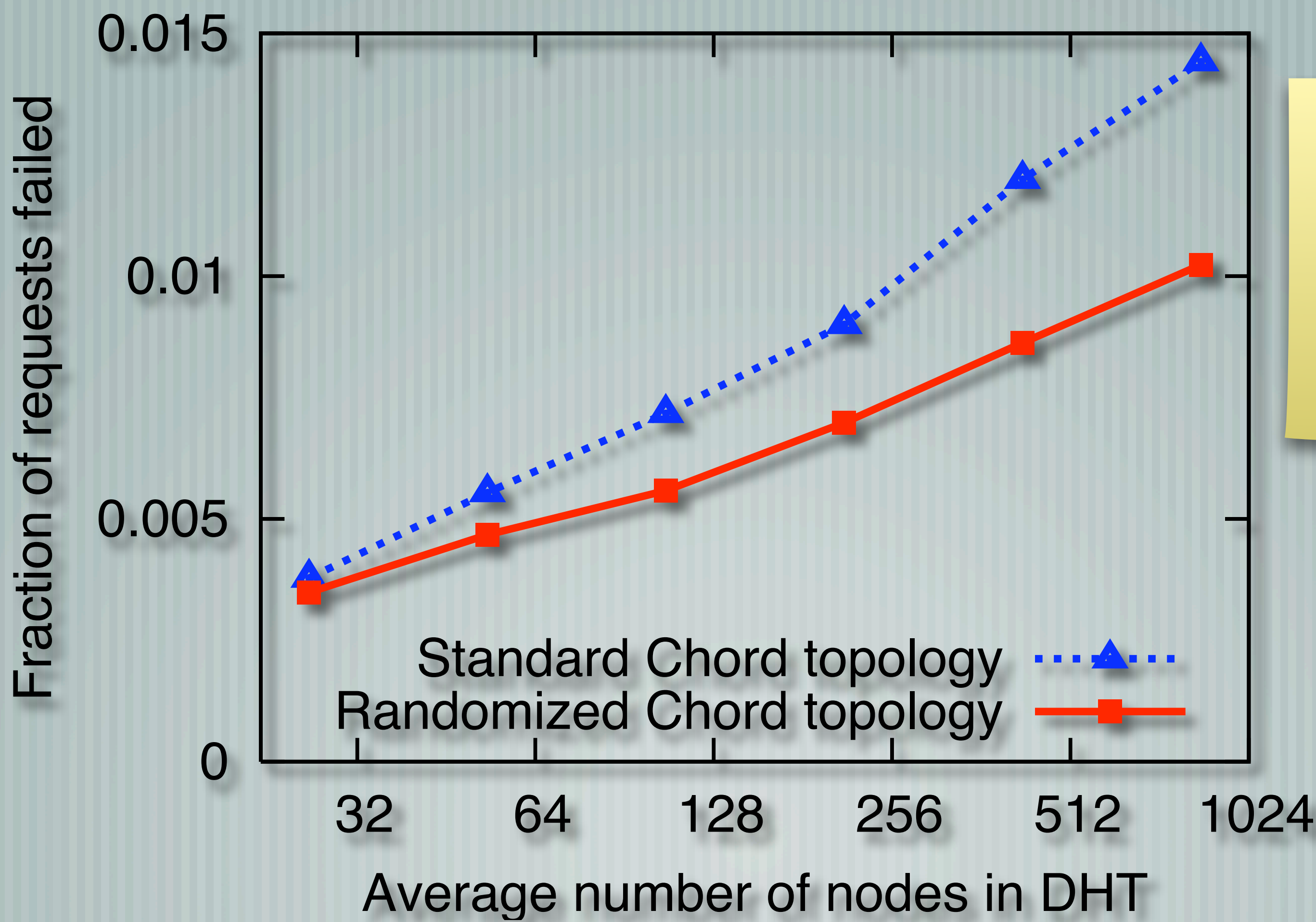
Randomized topology

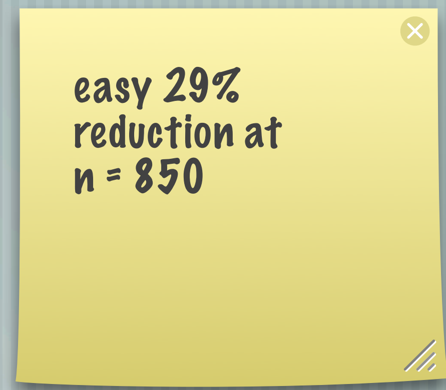


- Divide keyspace into 1/2, 1/4, 1/8, ...
- Finger points to random key within each interval

# DHT neighbor selection

## Datagram-level simulation, *i*3 Chord codebase, Gnutella trace



# contents

- an example system
- evaluation of node selection strategies

  *(how can we minimize churn?)*
- applications

  *(how do existing systems select nodes?)*
- **conclusions**

# conclusions

- A guide to minimizing churn
  - RR is pretty good; PL much worse
  - RR and PL arise in many systems
- Design insights
  - watch out for (implicit) PL strategies
  - easy way to reduce churn: add some randomness

doing less work may improve performance!

# backup slides

# Why use RR?

- Simplicity: no need to monitor and disseminate failure data
- Robustness to self-interested peers
- Legacy systems