Failure Recovery for Structured P2P Networks: Protocol Design and Performance under Churn*

Simon S. Lam and Huaiyu Liu

*includes results from version published in *Computer Networks* as well as TR-03-13

Structured P2P networks

- Of interest in this paper is the hypercube routing scheme used in PRR, Pastry and Tapestry
- Objective: Design protocols to construct and maintain consistent neighbor tables
- Question: *How high a rate of node dynamics can be supported?*

Outline

- The problem
- Overview of hypercube routing scheme
- Our approach
  - K-consistent network
  - Basic failure recovery
  - Join protocol for K-consistency
  - Protocol design for concurrent joins and failures
- Churn experiments
- Conclusions

Overview of Hypercube Routing Scheme

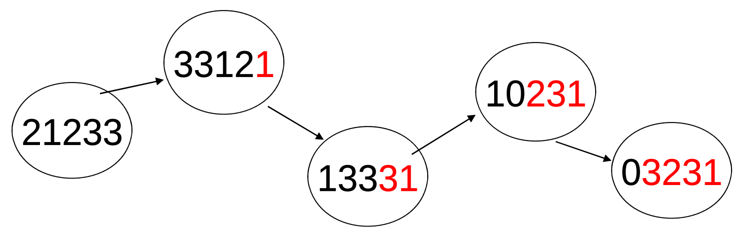
- Each node has an ID, a random fixed-length binary string, e.g., 128-bit MD5 hash of a name
  - concept of circular ID space
- Each node ID is represented by ***d*** digits of base ***b***, for example,

  0100111011 ➔ 10323 (d = 5, b = 4)
- We use suffix matching, as in PRR, with the rightmost digit being the 0^{th} digit

Routing Scheme

- Routing to a destination node is resolved digit by digit, trying to match at least one extra digit per hop

Example: source 21233, destination 03231



Neighbor Table at each node

- ***d*** levels, ***b*** entries at each level
- required suffix of (*i, j*)-entry in table of node *x* : *j* followed by the rightmost *i* digits in the node's ID

Example: neighbor table of node 21233 (***d***=5, ***b***=4)

Level 4	Level 3	Level 2	Level 1	Level 0
	10233	31033	22303	01100
11233	21233	03133	13113	33121
21233		21233	00123	12232
	03233		21233	21233

↓ *j*

← *i*

Neighbor Table at each node

- *d* levels, *b* entries at each level
- required suffix of (*i, j*)-entry in table of node *x* : *j* followed by the rightmost *i* digits in the node's ID

Example: neighbor table of node 21233 (***d***=5, ***b***=4)

Level 4	Level 3	Level 2	Level 1	Level 0
	10233	31033	22303	01100
11233	**21233**	03133	13113	33121
21233		**21233**	00123	12232
	03233		**21233**	**21233**

↓ *j*

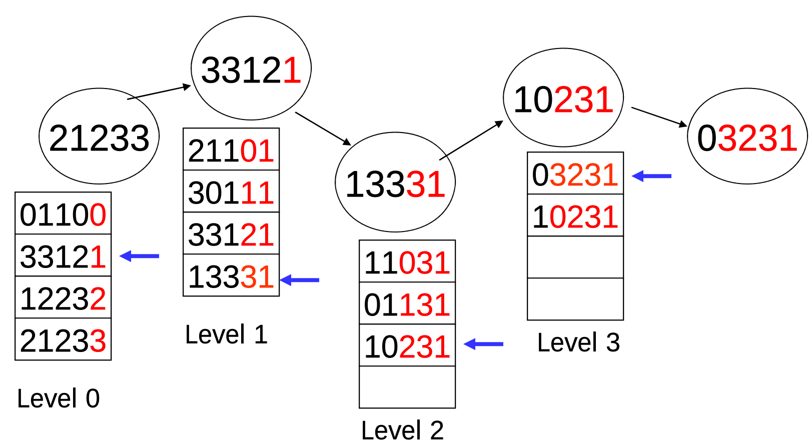
← *i*

Node *x* fills itself into *(i, x[i])* entries

Sigmetrics 2004 (Simon Lam) 7

Routing Scheme Revisited

- source 21233, destination 03231



Sigmetrics 2004 (Simon Lam) 8

Outline

- The problem
- Overview of hypercube routing scheme
- Our approach
  - K-consistent network
  - Basic failure recovery
  - Join protocol for K-consistency
  - Protocol design for concurrent joins and failures
- Churn experiments
- Conclusions

Consistency Definition

- A network is consistent iff for each table entry
  - if there exist nodes whose IDs have the required suffix of the entry, then the entry is filled with such a node (no false negative);
  - otherwise, the entry is empty (no false positive).

	01233	10233	0233	31033	033	22303	03	01100	0
11233	11233	21233	1233	03133	133	13113	13	33121	1
21233	21233		2233	21233	233	00123	23	12232	2
	31233	03233	3233		333	21233	33	21233	3

neighbor table of node 21233 (***d***=5, ***b***=4)

Consistency Property

- **Lemma** In a *consistent* network, every node is *reachable* from every other node.

Consistency can be broken by a single failure!

  - Note: No "false negative" is sufficient for reachability

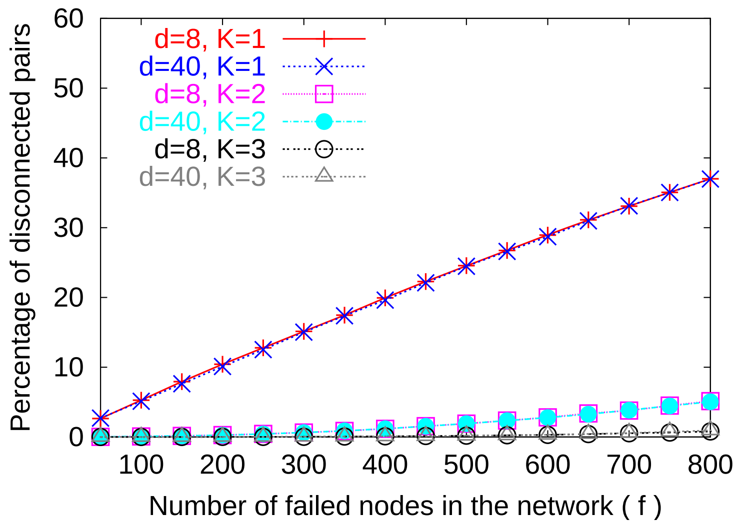
K-consistent Network: Definition

- A network is *K*-consistent iff: Every table entry stores *min(K,H)* neighbors, where *H* is the number of nodes with the required suffix of the entry

	10233	31033	22303 02203	01100 23310
11233	21233 11233	03133 10133	13113 00013	33121 10131
21233		21233 03233	00123 22323	23212 12232
	03233		21233 03133	00013 21233

Example: neighbor table of node 21233 for 2-consistency

K-consistent Network: routing redundancy



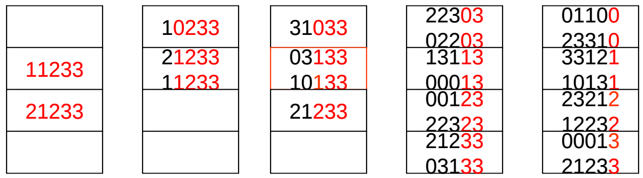
- Simulation results (n=4000, b=16)

Protocol design

- Objective: A K-consistent network under churn, for K > 1, is 1-consistent all the time
- Extend join protocol to build and maintain *K*-consistent neighbor tables, $K > 1$
  - generalize definitions of C-set tree template, C-set tree realization, and correctness conditions
  - extend join-noti level to *join-attach* level
- Failure recovery actions based upon each node's local info
  - a larger K is better (more neighbors)
  - a larger b is also better
- Integrate join and failure recovery protocols—how?

Join protocol example

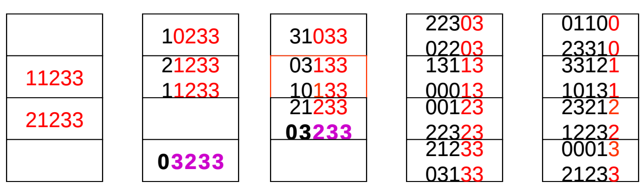
- Node 21233 with neighbor table



- A join-wait message from node 03233

Join protocol example (cont.)

- Node 21233 with neighbor table



- A join-wait message from node 03233
  - join-noti level is 3
  - join-attach level is 2

Basic Failure Recovery

- Assumption:
  - A network of n nodes, initially K-consistent
  - f out of n nodes fail (fail-stop)

- Goal: when failure recovery processes terminate
  - the network is K-consistent again
  - all "recoverable holes" are repaired (irrecoverable holes do not need repair)

- Difficulties
  - No global knowledge
  - Individual nodes do not know if a hole is "recoverable"

Using local information

- A node u is a *qualified substitute* of a failed node that has left a hole in a table entry if
  - u has the required suffix of the entry,
  - u not already in the entry
  - u has not failed
- In our protocol, each node maintains a *list of failed nodes* it has detected so far and uses it to determine if nodes can be used as qualified substitutes
  - a failed node needs to stay on the list for a time duration slightly larger than the probing period

Basic Failure Recovery Protocol

- A sequence of search steps, based on local information

Neighbor 2303 fails

1. Neighbors
2. Reverse neighbors
3. Failed nodes detected so far

0233	1033	~~2303~~ 2203	1100 3310
1233	3133 0133	3113 0013	3121 0131
	1233 3233	0123 2323	3212 2232
3233		1233 3133	0013 1233

Neighbor table of node 1233

STEP (a): search among neighbors and reverse-neighbors

Basic Failure Recovery Protocol

- A sequence of search steps, based on local information

Neighbor 2303 fails

1. Neighbors
2. Reverse neighbors
3. Failed nodes detected so far

0233	1033	~~2303~~ 2203	1100 3310
1233	3133 0133	3113 0013	3121 0131
	1233 3233	0123 2323	3212 2232
3233		1233 3133	0013 1233

Neighbor table of node 1123

STEP (b): query remaining neighbors in the same entry

(set up a timer to wait for replies)

Basic Failure Recovery Protocol

- A sequence of search steps, based on local information

Neighbor 2303 fails

1. Neighbors
2. Reverse neighbors
3. Failed nodes detected so far

0233	1033	~~2303~~ 2203	1100 3310
1233	3133 0133	3113 0013	3121 0131
	1233 3233	0123 2323	3212 2232
3233		1233 3133	0013 1233

Neighbor table of node 1123

STEP (c): query remaining neighbors at the same level

(set up a timer to wait for replies)

Basic Failure Recovery Protocol

- A sequence of search steps, based on local information

Neighbor 2303 fails

1. Neighbors
2. Reverse neighbors
3. Failed nodes detected so far

0233	1033	~~2303~~ 2203	1100 3310
1233	3133 0133	3113 0013	3121 0131
	1233 3233	0123 2323	3212 2232
3233		1233 3133	0013 1233

Neighbor table of node 1123

STEP (d): query all remaining neighbors

(set up a timer to wait for replies)

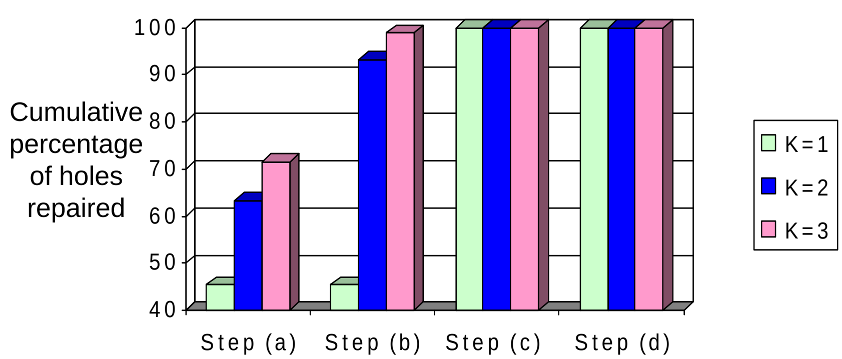
Failure Recovery is Effective

- 2,080 experiments, K=1 ~ 5, n=1000~8000
- 5% - 50% nodes fail, all nodes fail at the same time in majority of experiments
- All "recoverable holes" are repaired in every experiment, for K≥2

K, n	Number of simulations	Number of perfect recoveries	K, n	Number of simulations	Number of perfect recoveries
1,1000	100	51	1, 2000	180	96
2,1000	100	100	2, 2000	180	180
3,1000	100	100	3, 2000	180	180
4,1000	100	100	4, 2000	180	180
5,1000	100	100	5, 2000	180	180
1,4000	116	65	1, 8000	20	14
2,4000	116	116	2, 8000	20	20
3,4000	116	116	3, 8000	20	20
4,4000	116	116	4, 8000	20	20
5,4000	116	116	5, 8000	20	20

Failure Recovery is Efficient

- Majority of rec. holes repaired in step (a), no communication cost
- For K=2, 99.8% of all rec. holes repaired by step (c) with at most 2Kb messages for repairing a hole



Example: 800 out of 4000 nodes fail, b=16, d=40

Recoverable and Irrecoverable Holes

b, d, K	Total number of holes	Irreco-verable holes	Number of recoverable holes repaired at each step				
			step (a)	step (b)	step (c)	step (d)	not rec-overed
4, 64, 1	13125	1484	5257	0	5464	907	13
4, 64, 2	28616	3660	16675	6737	1496	48	0
4, 64, 3	43323	5798	28527	8613	339	46	0
4, 64, 4	57462	7997	40370	8988	70	37	0
4, 64, 5	70798	10174	51626	8945	37	16	0
16, 40, 1	29803	4442	11505	0	13833	23	0
16, 40, 2	55977	8161	30305	14301	3203	7	0
16, 40, 3	81406	9945	51203	19493	764	1	0
16, 40, 4	107547	10500	75028	21804	215	0	0
16, 40, 5	132257	10696	100157	21336	68	0	0

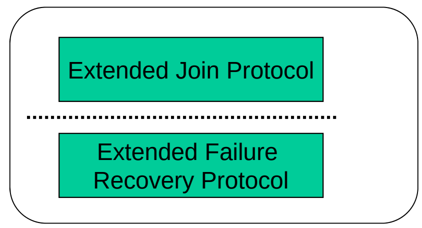
Table 4: Total number of holes, irrecoverable holes, and recoverable holes repaired at each step, $\boxed{n = 4000, f = 800}$

Join protocol for K-consistency

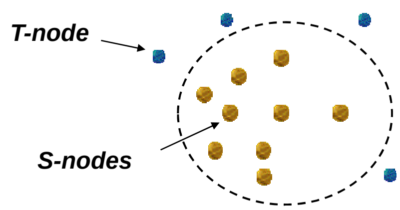
- Joining node copying, waiting, notifying, and in-system as before
- Concept of noti-level generalized to attach-level
  - Suppose node x sends JoinWaitMsg to node y which replies positively; *attach-level* is the lowest level node x is stored by node y
- Proved correct for an arbitrary sequence of concurrent joins in the absence of leaves/failures

Integrating Join and Failure Recovery Protocols

- Module compositon approach [LS 94]
- Extended join protocol assumes that failure recovery provides "perfect" recovery service
  - For each hole left by a failed neighbor, failure recovery returns with a qualified substitute within bounded delay; else, hole is irrecoverable
- Failure recovery actions are given higher priority than join actions to avoid circular reasoning



Protocol Extensions



- Failure recovery needs to distinguish *T-nodes* and *S-nodes*
  - To fill a hole, choose a S-node before a T-node
- Join protocol needs to be extended with the ability to invoke failure recovery and to backtrack
  - When a node detects a hole left by a failed neighbor, it starts an error recovery process *or backtracks* when certain conditions hold.
  - To fill a hole, choose a S-node before a T-node
  - When in failure recovery, delay processing join messages
  - When in failure recovery, a T-node cannot change its status to become S-node
  - (several more) …

Simulation Results

		$K = 1$		$K = 2, 3, 4, 5$	
n	No. of events ($\|W\| + \|F\|$)	No. of sim.	No. of sim. w/ perfect outcome	No. of sim.	No. of sim. w/ perfect outcome
1600	200 (38+162)	16	16	64	64
1600	200 (110+90)	16	16	64	64
1600	200 (160+40)	12	12	48	48
1600	400 (85+315)	12	10	48	48
1600	400 (204+196)	12	11	48	48
1600	400 (323+77)	12	12	48	48
1600	800 (386+414)	24	22	96	96
3600	400 (81+319)	16	13	64	64
3600	400 (210+190)	16	15	64	64
3600	400 (324+76)	12	12	48	48
3600	800 (169+631)	12	9	48	48
3600	800 (387+413)	12	11	48	48
3600	548 (400+148)	12	10	48	48
3200	1600 (780+820)	12	9	48	48

Table 5: Results for concurrent joins and failures

- 980 experiments, for n=3200, 3600, all joins and failures start at once
- Perfect outcome ~ all remaining nodes ($V \cup W - F$) satisfy *K*-consistency

Outline

- The problem
- Overview of hypercube routing scheme
- Our approach
  - K-consistent network
  - Basic failure recovery
  - Join protocol for K-consistency
  - Protocol design for concurrent joins and failures
- Churn experiments
- Conclusions

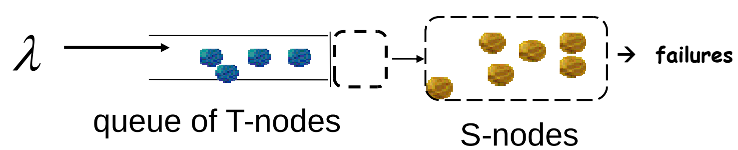
Churn Experiments

- How high a rate of node dynamics can be sustained?

- Start with a K-consistent network of 2000 nodes
- Generate join and failure events for 10,000 simulation seconds
  - join rate = failure rate = λ (churn rate)
- Take a snapshot every 50 seconds
  - evaluate connectivity and consistency measures
- Convergence to K-consistency at the end?

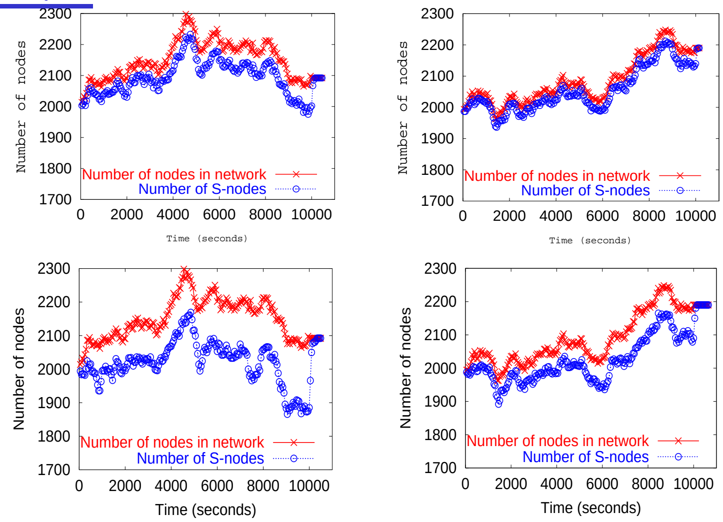
Observations

- Sustainable churn rate is upper bounded by the network's *join capacity*
- Join capacity: the rate at which new nodes can join the network successfully



- Limiting factors
  - K
  - failure rate λ
  - timeout value in each failure recovery step

Number of Nodes and S-nodes vs. Time



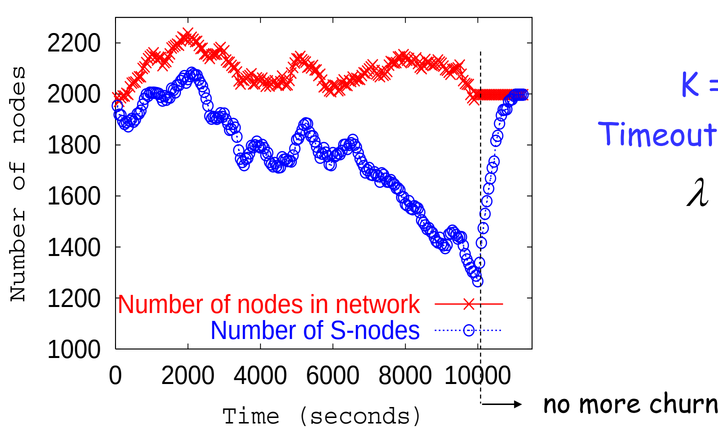
$\lambda = 1$

$\lambda = 1.5$

Timeout = 10 sec, K=3 Timeout = 5 sec, K=3

When Join Capacity is Exceeded

- Number of T-nodes keeps increasing
- Unable to restore K-consistency at the end



K = 3

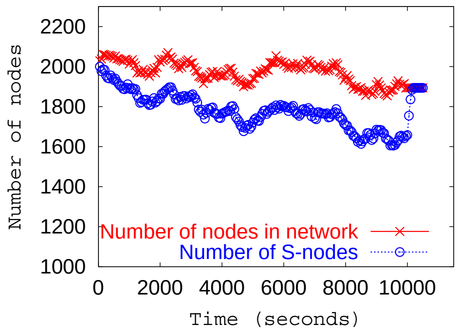
Timeout = 10sec

$\lambda = 2$

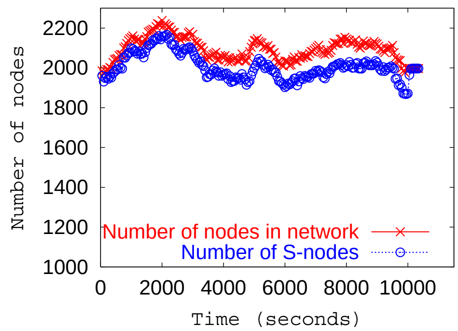
How to Increase Join Capacity?

- Choose a smaller K or a smaller timeout value

$$\lambda = 2$$

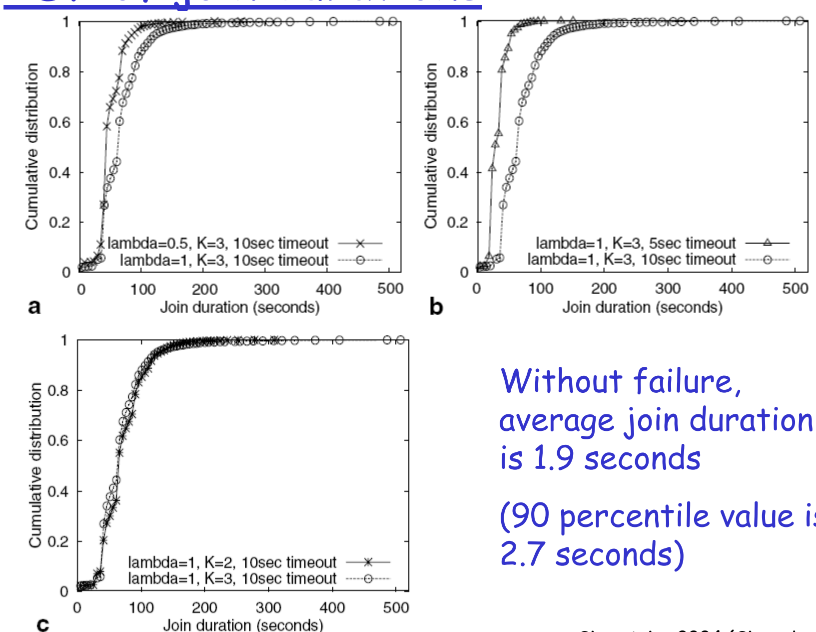


K=2, timeout = 10 sec



K=3, timeout = 5 sec

CDF of join durations



Without failure, average join duration is 1.9 seconds

(90 percentile value is 2.7 seconds)

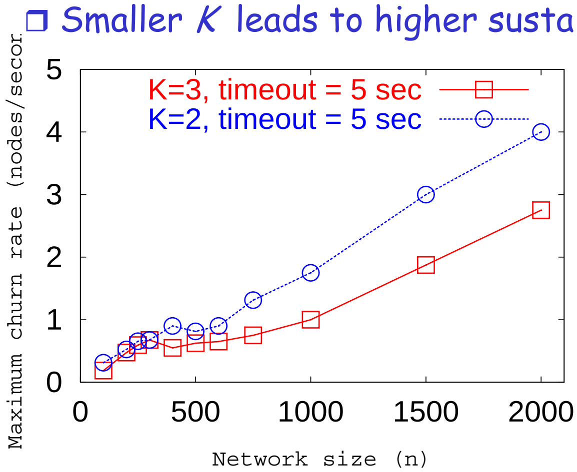
Summary of churn experiments

- n=2000, K=3, timeout=5 sec

λ	0.75	1	1.25	1.5	1.75	2
number of joins	7621	10080	12474	15011	17563	19957
number of failures	7423	9890	12468	14919	17563	19960
% snapshots, 3-con.-SAT	100	100	100	100	100	100
convergence to 3-con.	yes	yes	yes	yes	yes	yes
convergence time (sec.)	150	150	150	400	250	350
% snapshots, 1-con.	99.5	100	99.5	99	95.5	93
% snapshots, full connectivity	99.5	100	99.5	99.5	96.5	95
average %, connected s-d pairs	99.99999	100	99.99998	99.99998	99.99993	99.9997

Max Churn Rate vs. Network Size

- Max sustainable churn rate increases at least linearly with network size
- Smaller *K* leads to higher sustainable churn rate

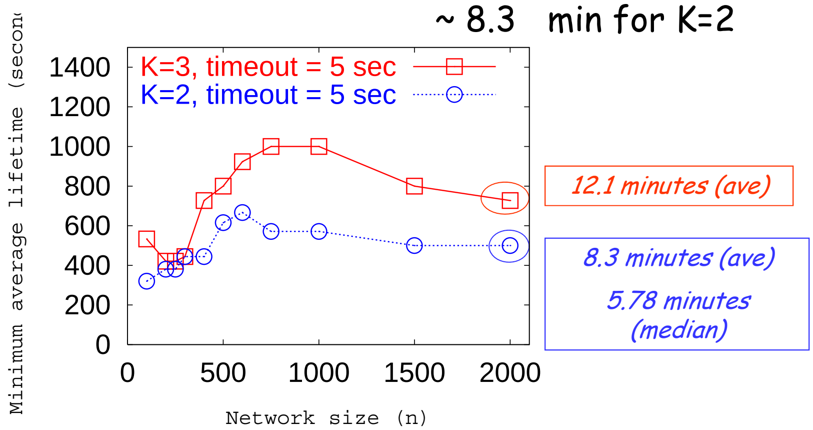


← $\lambda = 4$

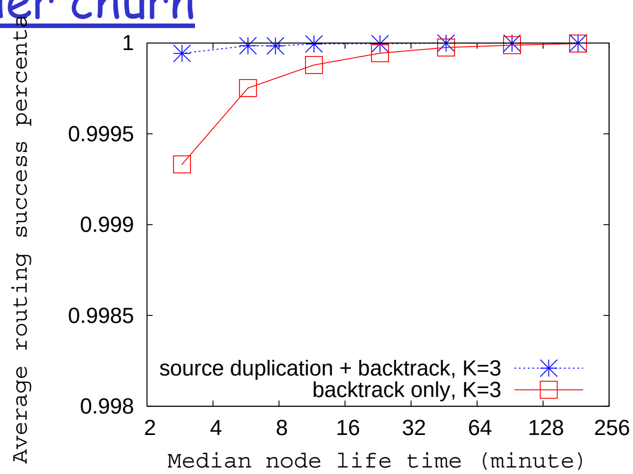
median node life time = **5.78 min**
(ave = 8.3 min)

Min Avg. Lifetime vs. Network Size

- The trend suggests:
  when n > 2000, avg. lifetime ~ 12.1 min for K=3,
  ~ 8.3 min for K=2

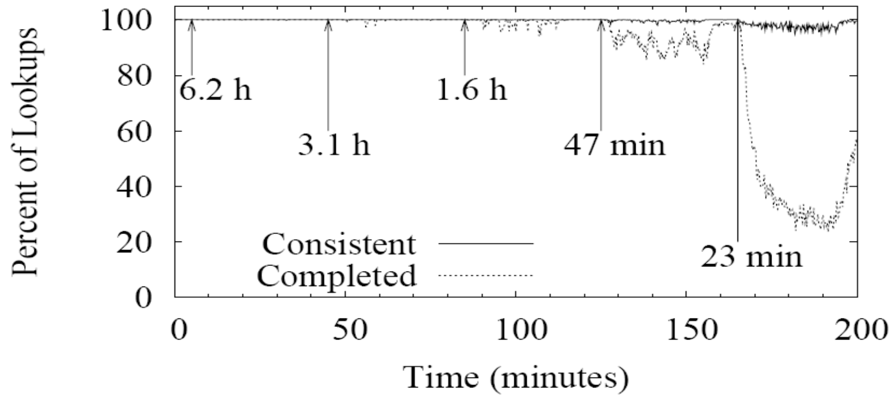


Successful routing % for systems under churn



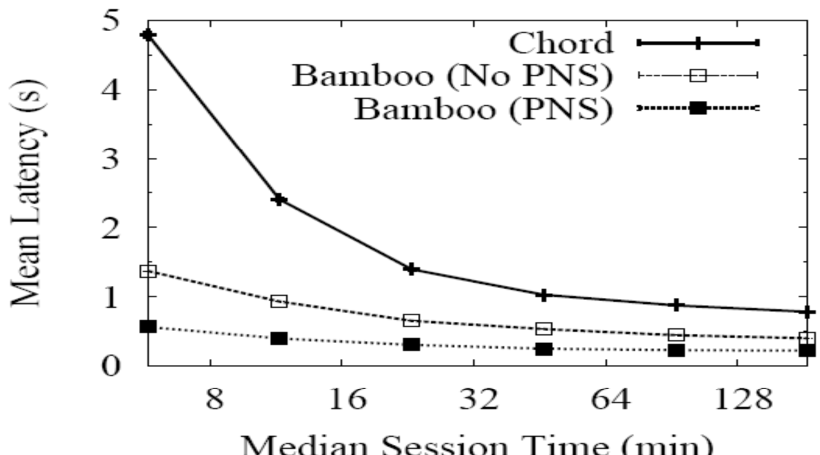
N = 2000, K=3, timeout = 2 sec

Pastry [Rhea et al. 2004]



- 1000 nodes Pastry network, each arrow indicates ave. lifetime
- incomplete -> routing terminated prior to destination
- "consistent" -> finds correct destination (if routing completed)

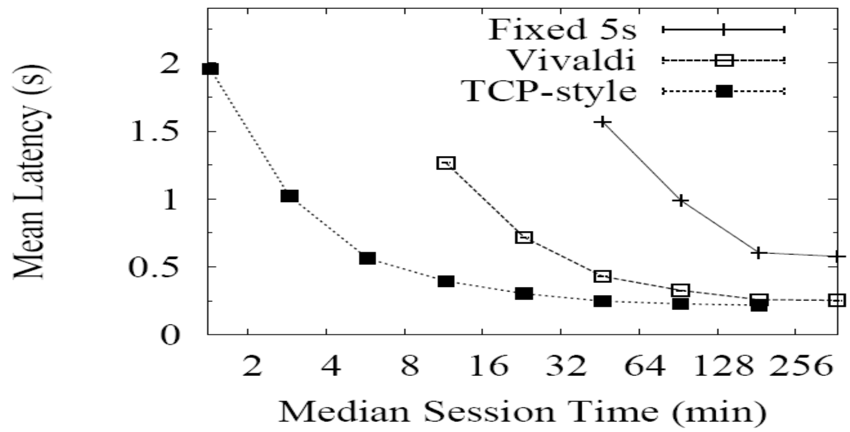
Chord vs. Bamboo [Rhea et al. 2004]



Bamboo is based on Pastry (paper is from Tapestry group!)

It uses proximity neighbor selection (PNS), better timeout estimates, periodic recovery (vs. reactive in Pastry)

Bamboo [Rhea et al. 2004]

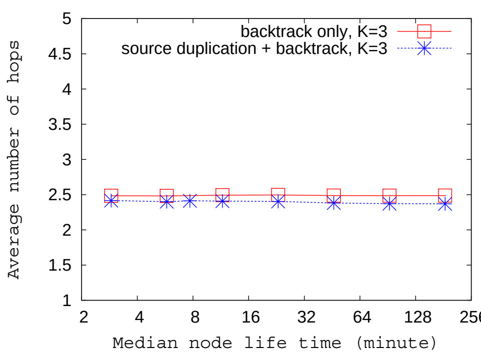


Better “timeout” estimate -> lower latency

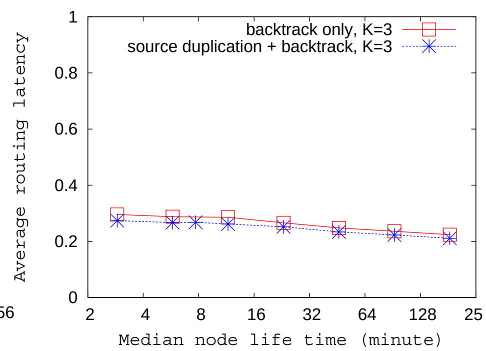
No mention of lookup completion and success rates

Our Hypercube Routing Performance

Average hop count



Average routing delay



n = 2000, K=3, timeout = 2 sec

Note that delay does not curve up when lifetime decreases <- because neighbor tables are consistent

Conclusions

- Introduced property of K-consistency for hypercube routing scheme
- Join and failure recovery protocols to maintain consistent neighbor tables under node dynamics
- The protocols are effective, efficient, and stable, for average node lifetime of a few minutes

Conclusions (cont.)

- Each network has a *join capacity* that
  - upper bounds its join rate
  - decreases when failure rate increases
  - can be increased by a smaller K or a smaller timeout value
- Recommended values for K:
  - for network with a high churn rate, K=2 or 3
  - for network with a low churn rate, K=3 or higher