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 $\rightarrow$ 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 ($d=5$, $b=4$)
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 ($d=5$, $b=4$)

<table>
<thead>
<tr>
<th>Level 4</th>
<th>Level 3</th>
<th>Level 2</th>
<th>Level 1</th>
<th>Level 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>11233</td>
<td>10233</td>
<td>31033</td>
<td>22303</td>
<td>01100</td>
</tr>
<tr>
<td>21233</td>
<td>21233</td>
<td>03133</td>
<td>13113</td>
<td>33121</td>
</tr>
<tr>
<td>03233</td>
<td>21233</td>
<td>21233</td>
<td>21233</td>
<td>21233</td>
</tr>
</tbody>
</table>

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

Routing Scheme Revisited

- Source 21233, destination 03231
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).

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

Example: neighbor table of node 21233 for 2-consistency
**K-consistent Network: routing redundancy**

![Graph showing percentage of disconnected pairs vs. number of failed nodes in the network](image)

- **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

```
  11233  21233  03233  23310  01100
  21233  11233  03133  10133  23310
  22303  02203  13113  33121  10133
  02203  13113  00013  22303  10133
  01100  23310  33121  10133  22303
```

- A join-wait message from node 03233

Join protocol example (cont.)

- Node 21233 with neighbor table

```
  11233  21233  03233  23310  01100
  21233  11233  03133  10133  23310
  22303  02203  13113  33121  10133
  02203  13113  00013  22303  10133
  01100  23310  33121  10133  22303
```

- 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
   - 0233
   - 1233
   - 3233

2. Reverse neighbors
   - 1033
   - 3133
   - 3233

3. Failed nodes detected so far
   - 2303
   - 1300
   - 3310

**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
   - 0233
   - 1233
   - 3233

2. Reverse neighbors
   - 1033
   - 3133
   - 3233

3. Failed nodes detected so far
   - 2303
   - 1300
   - 3310

**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

<table>
<thead>
<tr>
<th>Neighbors</th>
<th>Reverse Neighbors</th>
<th>Failed Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0233</td>
<td>1033 3133 1233</td>
<td>1100 3310</td>
</tr>
<tr>
<td>1233</td>
<td>3133 0133 3233</td>
<td>3113 3121</td>
</tr>
<tr>
<td>3233</td>
<td>1233 3233 0133</td>
<td>2323 2322</td>
</tr>
<tr>
<td></td>
<td>1233 3133 1233</td>
<td>0013 1233</td>
</tr>
</tbody>
</table>

Neighbor table of node 1123

**STEP (c):** query remaining neighbors at the same level
(set up a timer to wait for replies)

**STEP (d):** query all remaining neighbors
(set up a timer to wait for replies)
**Failure Recovery is Effective**

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

<table>
<thead>
<tr>
<th>$K, n$</th>
<th>Number of simulations</th>
<th>Number of perfect recoveries</th>
<th>$K, n$</th>
<th>Number of simulations</th>
<th>Number of perfect recoveries</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,100</td>
<td>100</td>
<td>31</td>
<td>1,200</td>
<td>180</td>
<td>26</td>
</tr>
<tr>
<td>2,100</td>
<td>100</td>
<td>100</td>
<td>2,200</td>
<td>180</td>
<td>180</td>
</tr>
<tr>
<td>3,100</td>
<td>100</td>
<td>100</td>
<td>3,200</td>
<td>180</td>
<td>180</td>
</tr>
<tr>
<td>4,1000</td>
<td>100</td>
<td>100</td>
<td>4,2000</td>
<td>180</td>
<td>180</td>
</tr>
<tr>
<td>5,1000</td>
<td>100</td>
<td>100</td>
<td>5,2000</td>
<td>180</td>
<td>180</td>
</tr>
<tr>
<td>1,4000</td>
<td>116</td>
<td>65</td>
<td>1,8000</td>
<td>20</td>
<td>14</td>
</tr>
<tr>
<td>2,4000</td>
<td>116</td>
<td>116</td>
<td>2,8000</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>3,4000</td>
<td>116</td>
<td>116</td>
<td>3,8000</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>4,4000</td>
<td>116</td>
<td>116</td>
<td>4,8000</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>5,4000</td>
<td>116</td>
<td>116</td>
<td>5,8000</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

**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

![Cumulative percentage of holes repaired](image)

---

Example: 800 out of 4000 nodes fail, $b=16$, $d=40$
### Recoverable and Irrecoverable Holes

<table>
<thead>
<tr>
<th>$b, d, K$</th>
<th>Total number of holes</th>
<th>Irrecoverable holes</th>
<th>Number of recoverable holes repaired at each step</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>step (a)</td>
<td>step (b)</td>
<td>step (c)</td>
</tr>
<tr>
<td>4, 64, 1</td>
<td>13125</td>
<td>1484</td>
<td>5257</td>
</tr>
<tr>
<td>4, 64, 2</td>
<td>28616</td>
<td>3660</td>
<td>16675</td>
</tr>
<tr>
<td>4, 64, 3</td>
<td>43323</td>
<td>5798</td>
<td>28527</td>
</tr>
<tr>
<td>4, 64, 4</td>
<td>57462</td>
<td>7997</td>
<td>40370</td>
</tr>
<tr>
<td>4, 64, 5</td>
<td>70798</td>
<td>10174</td>
<td>51626</td>
</tr>
<tr>
<td>16, 40, 1</td>
<td>29803</td>
<td>4442</td>
<td>11505</td>
</tr>
<tr>
<td>16, 40, 2</td>
<td>55977</td>
<td>8161</td>
<td>30305</td>
</tr>
<tr>
<td>16, 40, 3</td>
<td>81406</td>
<td>9945</td>
<td>51203</td>
</tr>
<tr>
<td>16, 40, 4</td>
<td>107547</td>
<td>10500</td>
<td>75028</td>
</tr>
<tr>
<td>16, 40, 5</td>
<td>132257</td>
<td>10696</td>
<td>100157</td>
</tr>
</tbody>
</table>

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

---

### 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 composition 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**

| n   | No. of events (|W| + |F|) | K = 1 | No. of sim. | No. of sim. w/ perfect outcome | K = 2, 3, 4, 5 | No. of sim. | No. of sim. w/ perfect outcome |
|-----|----------------|-------|--------|-------------|-------------------------------|----------------|-------------|-------------------------------|
| 1600| 200 (381+162)  | 16    | 16     | 64          | 64                            |                |             |                               |
| 1600| 200 (110+90)   | 16    | 16     | 64          | 64                            |                |             |                               |
| 1600| 500 (160+40)   | 15    | 15     | 48          | 48                            |                |             |                               |
| 1600| 400 (85+315)   | 12    | 10     | 48          | 48                            |                |             |                               |
| 1600| 400 (204+196)  | 12    | 11     | 48          | 48                            |                |             |                               |
| 1600| 400 (325+777)  | 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 U 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 = $\lambda$ (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 $\lambda$
- Timeout value in each failure recovery step
Number of Nodes and S-nodes vs. Time

![Graphs showing number of nodes and S-nodes over time for different values of \( \lambda \).]

- \( \lambda = 1 \)
- \( \lambda = 1.5 \)
- 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

![Graph showing number of nodes and S-nodes for \( K=3 \), \( \lambda = 2 \), and Timeout = 10 sec.]

- \( K = 3 \)
- Timeout = 10 sec
- \( \lambda = 2 \)
- no more churn
How to Increase Join Capacity?

- Choose a smaller $K$ or a smaller timeout value

$$\lambda = 2$$

**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

<table>
<thead>
<tr>
<th>λ</th>
<th>0.75</th>
<th>1</th>
<th>1.25</th>
<th>1.5</th>
<th>1.75</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of joins</td>
<td>7621</td>
<td>10080</td>
<td>12474</td>
<td>15011</td>
<td>17563</td>
<td>19957</td>
</tr>
<tr>
<td>number of failures</td>
<td>7423</td>
<td>9890</td>
<td>12468</td>
<td>14919</td>
<td>17563</td>
<td>19960</td>
</tr>
<tr>
<td>% snapshots, 3-con.-SAT</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>convergence to 3-con.</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>convergence time (sec.)</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>400</td>
<td>250</td>
<td>350</td>
</tr>
<tr>
<td>% snapshots, 1-con.</td>
<td>99.5</td>
<td>100</td>
<td>99.5</td>
<td>99</td>
<td>95.5</td>
<td>93</td>
</tr>
<tr>
<td>% snapshots, full connectivity</td>
<td>99.5</td>
<td>100</td>
<td>99.5</td>
<td>99.5</td>
<td>96.5</td>
<td>95</td>
</tr>
</tbody>
</table>

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

![Graph showing the relationship between network size and maximum churn rate with different values of λ and K.](Sigmetrics 2004 (Simon Lam) 37)

- K=3, timeout = 5 sec
- K=2, timeout = 5 sec

K=3, timeout = 5 sec
K=2, timeout = 5 sec

λ = 4

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

![Graph showing median and average node life with different values of K and λ.](Sigmetrics 2004 (Simon Lam) 38)
The trend suggests:
when \( n > 2000 \), avg. lifetime \( \sim 12.1 \) min for \( K=3 \),
\( \sim 8.3 \) min for \( K=2 \).

**Successful routing % for systems under churn**

\( N = 2000, K=3, \text{timeout} = 2 \text{ sec} \)
**Pastry** [Rhea et al. 2004]

- 1000 nodes Pastry network, each arrow indicates avg. 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**

<table>
<thead>
<tr>
<th>Average hop count</th>
<th>Average routing delay</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **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