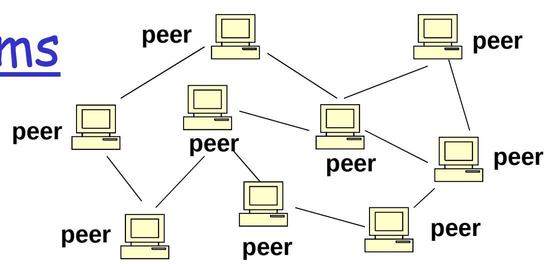
Neighbor Table Construction and Update in a Dynamic Peer-to-Peer Network

Huaiyu Liu and Simon S. Lam

P2P systems



- User machines (peers) cooperate to share resources
  - *Unstructured* systems: scoped flooding (e.g., original Gnutella)
  - Hierarchical systems: infrastructure nodes (e.g., BitTorrent)
  - Many copies of each object (file) in network
- O*verlay* networks that provide services
  - *Structured* p2p systems: PRR, Chord, Pastry, Tapestry, etc.
  - Routing tables provide more efficient routing
  - DHT applications
  - Performance impacted by churn

Hypercube routing scheme

- Routing infrastructure proposed in PRR [Plaxton et al 1997],
  - used in Pastry [2001], Tapestry [2001]
- In basic scheme, each node maintains a neighbor table, pointing to O(log *n*) nodes
  - O(log *n*) routing hops on the average
- PRR assumes static neighbor tables that are *consistent* and *optimal*
  - PRR guarantees locating a copy of a replicated object, if it exists, with asymptotically optimal cost

Talk Outline

- Overview of hypercube routing scheme
- Motivation and related work
- Conceptual foundation
- Join protocol
- Protocol analysis
- Conclusion

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,

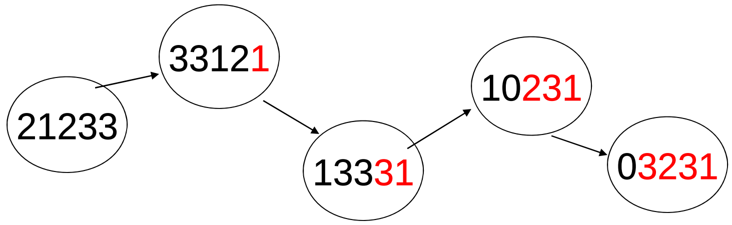
  0100111011 ➔ 10323 (***d*** = 5, ***b*** = 4)

- 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)

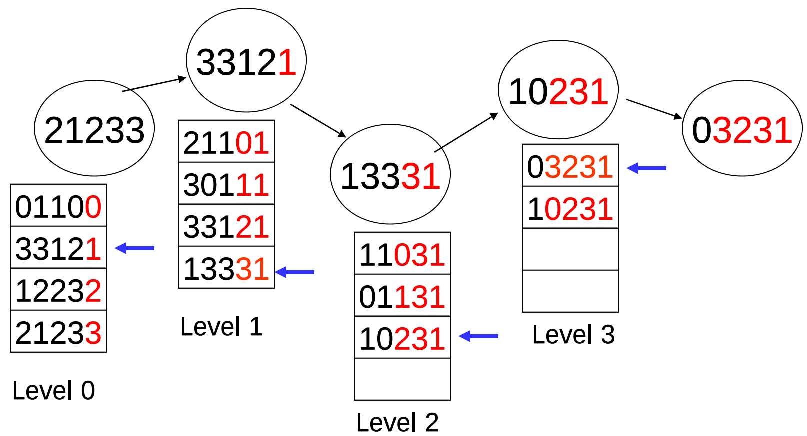
Level 4	Level 3	Level 2	Level 1	Level 0
	10233	31033	22303	01100
11233	**21233**	03133	13113	33121
21233		**21233**	00123	12232
	03233		**21233**	**21233**

↓ *j* ← *i*

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

Routing Scheme Revisited

- source 21233, destination 03231



Talk Outline

- Overview of hypercube routing scheme
- Motivation and related work
- Conceptual foundation
- Join protocol
- Protocol analysis
- Conclusion

Motivation—Protocols needed for Dynamic Networks

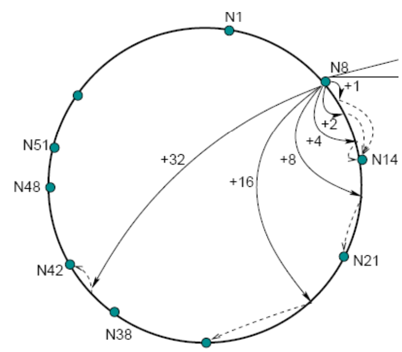
- To handle joins, leaves and failures
- Network initialization
- Neighbor table optimization

- Our objective:

  Protocols to construct consistent neighbor tables and maintain consistency under node dynamics

Related Work—Chord [2001]

- Not hypercube routing, but similar in spirit
- Each node keeps
  - successor and predecessor pointers form a ring
  - "finger pointers" provide short cuts
- Stabilization protocol to keep successor pointers up to date to guarantee "correctness"
  - maintaining consistency of finger pointers considered hard



Related Work—Pastry [2001]

- Each node also maintains a Leaf set of L nearest neighbors on the ID ring, e.g., L=32
- If the destination of a packet is within range of Leaf set, it is forwarded to its closest node in Leaf set; else, it is forwarding by hypercube routing
  - Rare case – forward packet to another node with the same prefix match as current node, but numerically closer to destination
- Pointers for hypercube routing are repaired "lazily"; emphasis on maintaining Leaf set for resilience

Related Work—Tapestry [2001]

- Object location—need a method to determine a single "root" node that matches with the longest prefix (or suffix) of an object
  - In a Tapestry node, when there is no match for the next digit of a packet, it is forwarded to the next filled entry at the same level in the routing table (wrapped around if necessary). It is proved that the node is unique.

Related Work—Tapestry [2002]

- A correctness proof for its join protocol based on
  - a lower-layer protocol for a joining node to send acknowledged multicast to all existing nodes with a given prefix
  - concurrent joins—pointer to a new node is locked after its multicast is received, and unlocked when all acks return from multicasts triggered by the new node

Finding hay versus finding needles

- For object location applications

When an object has many replicas in a network, the probability of finding one of them is high even when routing tables are far from being consistent

Contributions of this Paper

- A foundation, ***C-set trees***, for protocol design and reasoning about consistency
- New *join protocol* for hypercube routing
  - Proof by induction that the join protocol maintains consistency for an *arbitrary number of concurrent joins*
  - Join protocol can also be used for *network initialization*
  - Each joining node handles its own join process—no need for other nodes to maintain state information for joining nodes (no multicast, no locking)

Talk Outline

- Overview of hypercube routing scheme
- Motivation and related work
- Conceptual foundation
- Join protocol
- Protocol analysis
- Conclusion

Definition

- A *consistent* network:

  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; otherwise, the entry is empty.

31033	033
03133	133
21233	233
	333

Level 2, node 21233

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

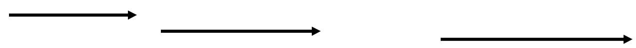
Assumptions and Goal

- Assumptions: When node x joins a network $\langle V, N(V) \rangle$
  - $V \neq \emptyset$ and $N(V)$ is *consistent*
  - x knows a node in V
  - messages are delivered reliably
  - no node failure or leave
- Goal: Construct tables of new nodes and update tables of existing nodes so that eventually, the new network is *consistent* again.

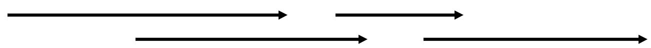
Definitions

- *Joining period* of a node.

- *Sequential* joins



- *Concurrent* joins



Notification set of x regarding V

Example: initial network
V={33121, 12100, 23133, 10033, 03213},
then 21233 and 02101 join

- The noti set of 21233 is {23133, 10033}
- The noti set of 02101 is {33121}

Definitions (cont.)

- *Independent* joins: for every pair of nodes in set W of joining nodes, their noti-sets are disjoint
  - Example: initial network V={33121, 12100, 23133, 10033, 03213}, then 21233 and 02101 join.

$$V_{21233}^{Notify} \cap V_{02101}^{Notify} = \varnothing$$

- *Dependent* joins (definition in paper):
  Example: 21233 and 00233 join the above network
  - Also the joins of x and y are dependent if there exists a joining node *u* such that *x*'s and *y*'s noti sets are subsets of *u*'s noti set
- Handling *concurrent* and *dependent* joins is the most difficult part.

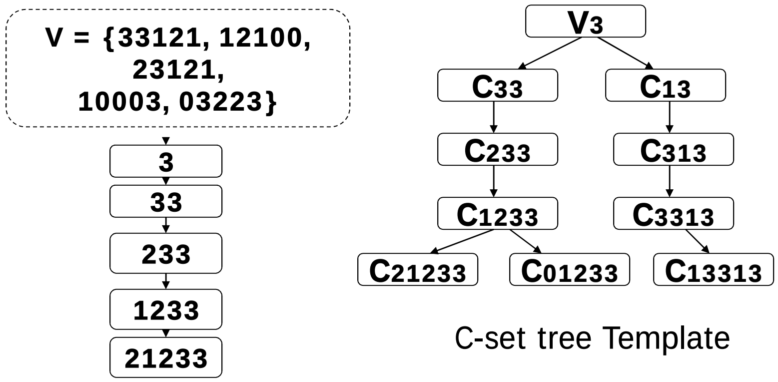
Goals of join protocol

Starting with a consistent network, <V, N(V)>, and a set W of joining nodes, the protocol goals are:

1. For $x \in W, y \in V$, eventually x and y can reach each other
2. For $x_1 \in W, x_2 \in W$, eventually, x_1 and x_2 can reach each other

C-set Tree

- A *conceptual structure* that guides our protocol design and proofs (not in implementation)



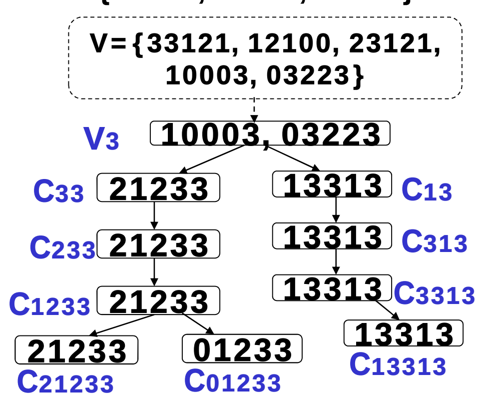
C-set tree Template

W = {21233, 01233, 13313}

C-set Tree (cont.)

- By filling new nodes into neighbor tables, the C-set tree is conceptually realized.
- Different sequences of message exchanges between nodes result in different realizations.

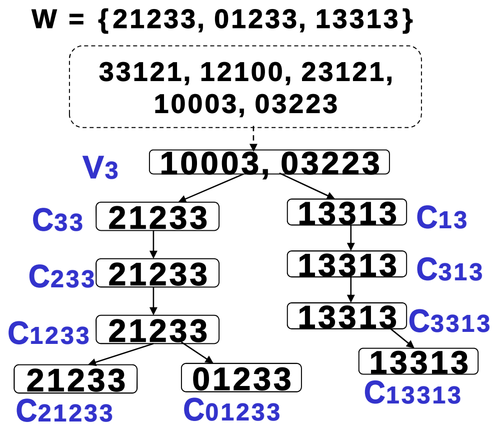
W = {21233, 01233, 13313}



C-set tree realization

C-set tree realization: Correctness Conditions

- Template and tree have same structure; no C-set is empty
- For each node *y* in root, for each child C-set of root, *y* stores a node with the required suffix of each child C-set
- For each leaf node x in tree, if a C-set along its path to root has a sibling, x stores a node with the suffix of the sibling



C-set tree Realization

More details …

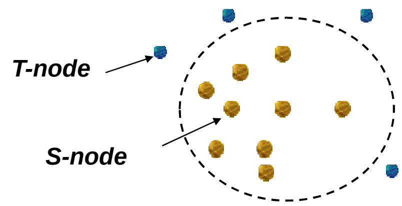
- For independent joins, their noti-sets in V are disjoint – therefore, no need to know about each other
- For concurrent joins in general, the noti-sets may be different for different subsets of nodes in W, there are two cases (Proposition 5.5):
  - the noti-sets are disjoint
  - one noti-set is a proper subset of the other

Talk Outline

- Overview of hypercube routing scheme
- Motivation and related work
- Conceptual foundation
- Join protocol
- Protocol analysis
- Conclusion

Join Protocol: Intuition

- T-nodes and S-nodes
  - T: nodes joining a network
  - S: nodes that finished joining



- T-node needs to:
  - copy neighbors from S-nodes
  - find a position for itself in the C-set tree (find a S-node to store it as a neighbor)
  - find and notify others in the same tree

Join Protocol

- Status of a joining node: *copying, waiting, notifying, in_system*

copying : Copies and constructs neighbor table level by level

waiting : Attaches itself to the network, i.e., finds an S-node to store it as a neighbor

notifying : Searches and notifies nodes with a certain suffix

in_system: Becomes an S-node

Join Protocol

- Status of a joining node: *copying, waiting, notifying, in_system*

copying : Copies and constructs neighbor table level by level

waiting : Attaches itself to the network, i.e., finds an S-node to store it as a neighbor (common suffix is its noti-level)

notifying : Searches and notifies nodes with a certain suffix

in_system: Becomes an S-node

Join Protocol

- Status of a joining node: *copying, waiting, notifying, in_system*

copying : Copies and constructs neighbor table level by level

waiting : Attaches itself to the network, i.e., finds an S-node to store it as a neighbor

notifying : Searches and notifies nodes with a common suffix of length ≥ its noti-level

in_system: Becomes an S-node

Join Protocol

- Status of a joining node: *copying, waiting, notifying, in_system*

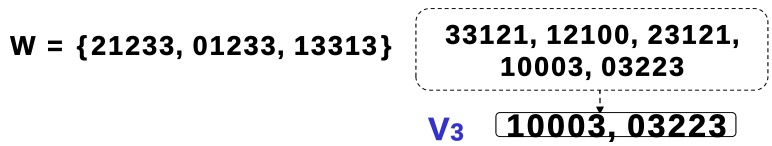
copying : Copies and constructs neighbor table level by level

waiting : Attaches itself to the network (i.e., finds an S-node to store it as a neighbor)

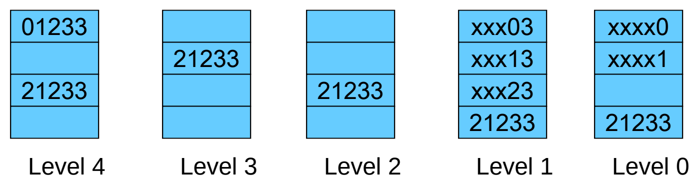
notifying : Searches and notifies nodes with a certain suffix

in_system: Becomes an S-node, replies to pending JoinWait requests, informs all of its reverse neighbors

Join Protocol: An Example

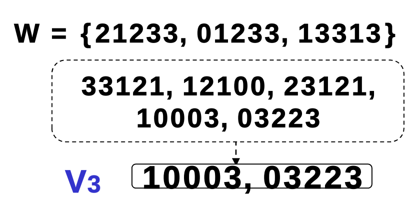


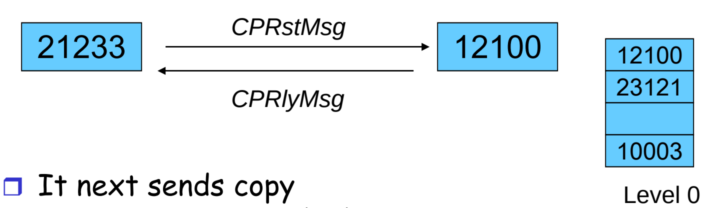
After the joins, from global info, neighbor table of 21233 should look like



Join Protocol: An Example

- 21233 knows 12100
- *copying* : 21233 copies and constructs neighbor table level by level

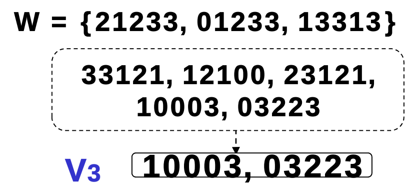


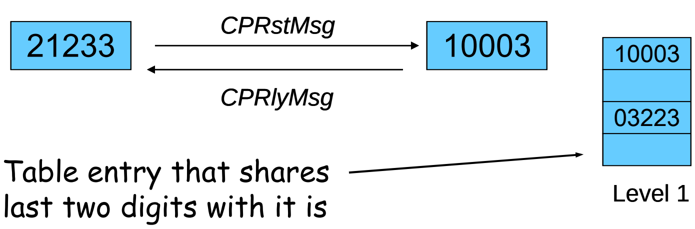


- It next sends copy request to 10003 which shares last digit with it

Join Protocol: An Example (cont.)

- ***copying*** : 21233 copies and constructs neighbor table level by level

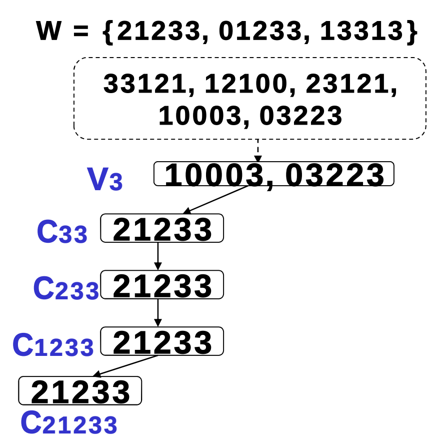


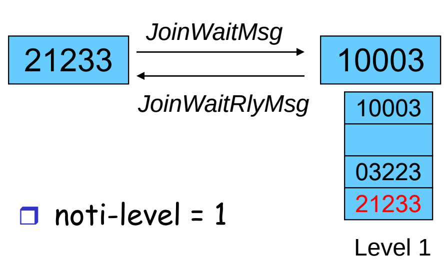


- Table entry that shares last two digits with it is empty => change status to waiting and ask 10003 to store it as a neighbor

Join Protocol: An Example (cont.)

- ***waiting*** : 21233 tries to attach itself to the network (i.e., to find an S-node to store it as a neighbor)

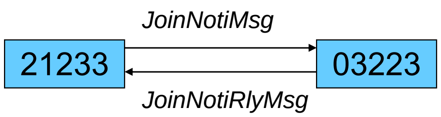


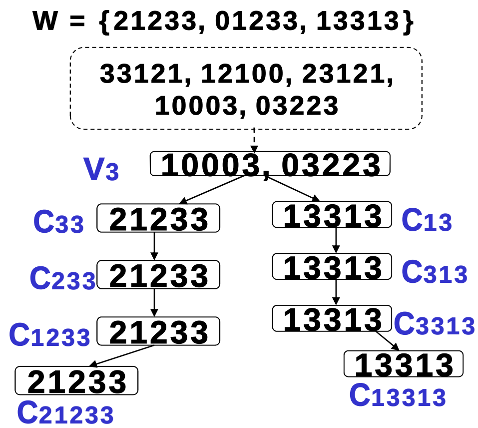


- noti-level = 1

Join Protocol: An Example (cont.)

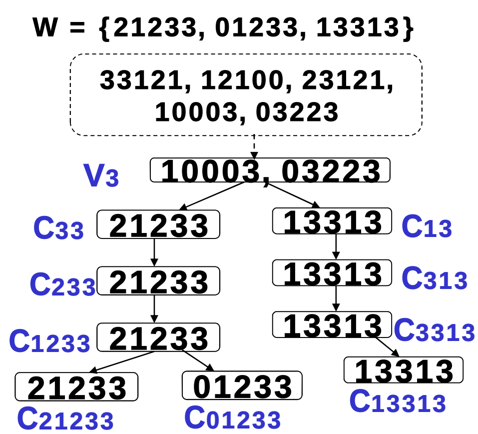
- ***notifying*** : 21233 searches and notifies nodes with a common suffix ≥ xxxx3





Join Protocol: An Example (cont.)

- ***notifying*** : 21233 learns about 01233 and 13313 through 10003 or 03223 or vice versa
- ***in_system***: Becomes an S-node



Join Protocol: An Example (cont.)

W = {21233, 01233, 13313}

33121, 12100, 23121, 10003, 03223

V_3 10003, 03223

After the joins, routing table of 21233 is possibly as shown below

Level 4	Level 3	Level 2	Level 1	Level 0
01233			10003	12100
	21233		13313	23121
21233		21233	03223	
			21233	21233

Note: on the average, only O($\log_d$ n) levels need to be stored

State variables of a joining node x

$x.status \in \{copying, waiting, notifying, in\_system\}$, initially *copying*
$N_x(i,j)$: the (i,j)-neighbor of x, initially *null*.
$N_x(i,j)$.state $\in \{T, S\}$.
$R_x(i,j)$: the set of reverse(i,j)-neighbors of x, initially *empty*.

$x.noti\_level$: an integer, initially 0.
Q_r: a set of nodes from which x waits for replies, initially *empty*.
Q_n: a set of nodes x has sent notifications to, initially *empty*.
Q_j: a set of nodes that have sent x a *JoinWaitMsg*, initially *empty*.
Q_{sr}, Q_{sn}: a set of nodes, initially *empty*.

Protocol messages

CpRstMsg, sent by x to request a copy of receiver's neighbor table.
CpRlyMsg($x.table$), sent by x in response to a *CpRstMsg*.
JoinWaitMsg, sent by x to notify receiver of the existence of x, when $x.status$ is *waiting*.
JoinWaitRlyMsg($r, u, x.table$), sent by x in response to a *JoinWaitMsg*, $r \in \{negative, positive\}$, u: a node.
JoinNotiMsg($x.table$), sent by x to notify receiver of the existence of x, when $x.status$ is *notifying*.
JoinNotiRlyMsg($r, x.table, f$), sent by x in response to a *JoinNotiMsg*, $r \in \{negative, positive\}$, $f \in \{true, false\}$.
InSysNotiMsg, sent by x when $x.status$ changes to *in_system*.
SpeNotiMsg(x, y), sent or forwarded by a node to inform receiver of the existence of y, where x is the initial sender.
SpeNotiRlyMsg(x, y), response to a *SpeNotiMsg*.
RvNghNotiMsg(y, s), sent by x to notify y that x is a reverse neighbor of y, $s \in \{T, S\}$.
RvNghNotiRlyMsg(s), sent by x in response to a *RvNghNotiMsg*, $s = S$ if $x.status$ is *in_system*; otherwise $s = T$.

Talk Outline

- Overview of hypercube routing scheme
- Motivation and related work
- Conceptual foundation
- Join protocol
- Protocol analysis
  - assuming reliable message delivery, no node deletion
- Conclusion

Protocol Analysis: Correctness

- *Consistency*

Theorem 1 *Suppose a set of nodes,* $W = \{x_1, ..., x_m\}$, $m \geq 1$, *join a consistent network* $\langle V, \mathcal{N}(V) \rangle$. *Then, at time* t^e, $\langle V \cup W, \mathcal{N}(V \cup W) \rangle$ *is consistent.*

- *Termination*

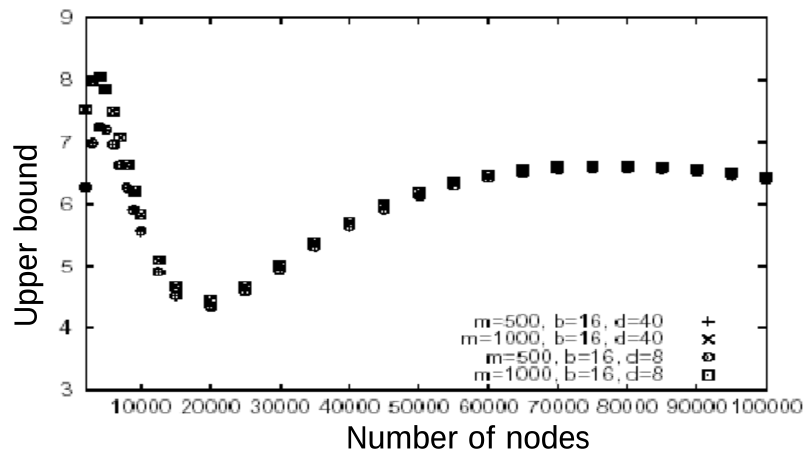
Theorem 2 *Suppose a set of nodes,* $W = \{x_1, ..., x_m\}$, $m \geq 1$, *join a consistent network* $\langle V, \mathcal{N}(V) \rangle$. *Then, each node* x, $x \in W$, *eventually becomes an S-node.*

Protocol Analysis: Communication Cost

- The number of CpRstMsg and JoinWaitMsg messages sent by a joining node during status *copying* and *waiting* is at most *d+1* (**Theorem 3**).
- An upper bound on the expected number of JoinNotiMsg messages sent during the *notifying* status by a joining node (**Theorem 5**).
- These three messages and their replies are large because each such message/reply may contain a neighbor table.

Communication Cost (cont.)

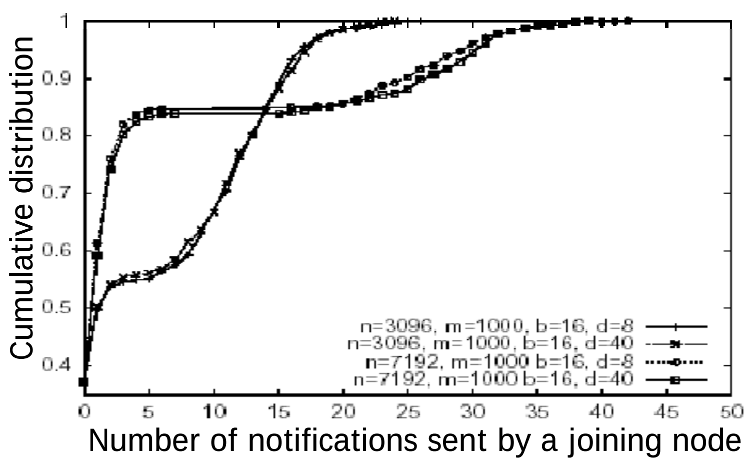
- Upper bound on expected no. of notifications (from Theorem 5)



ICDCS 2003 (Simon Lam) 47

Communication Cost (cont.)

- From simulations – 1000 nodes concurrently join 3096 nodes, 1000 nodes concurrently join 7192 nodes
  - all joins start at the same time



ICDCS 2003 (Simon Lam) 48

Comparing theoretical and simulation results

- For the four simulation cases, the average number of join notification messages sent

simulations	analytic upper bound
6.12	8.00
6.05	8.00
5.03	6.99
5.40	6.99

Network initialization

- The join protocol can be used to build consistent neighbor tables for a set of n nodes.
  - put one node x in V with x.table filled in as follows:
    - $N_x(i, x[i]) = x$, $N_x(i, x[i]).state = S$, $i \in [d]$.
    - $N_x(i, j) = null$, $i \in [d]$, $j \in [b]$ and $j \neq x[i]$.
  - Given x, the other n-1 nodes join the network concurrently.

Conclusions

- A new join protocol for hypercube routing scheme
  - for concurrent joins
  - each joining node maintains state info for its own join process
- A conceptual structure, C-set trees, for reasoning about consistency
  - a guide for protocol design and proof construction
- Proved that join protocol constructs and maintains consistent neighbor tables for any number of concurrent joins (in the absence of node leave or failure).
  - Join processes terminate under standard assumptions
- Analyzed communication costs
- Protocols for leaves and failures—next paper

End