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

Huaiyu Liu and Simon S. Lam

Department of Computer Sciences,
University of Texas at Austin,
Austin, TX 78712
{huaiyu, lam}@cs.utexas.edu

TR-02-46 September 20, 2002
Revised, February 2003

Abstract

In a system proposed by Plaxton, Rajaraman and Richa (PRR), the expected cost of accessing a replicated object was proved to be asymptotically optimal for a static set of nodes and pre-existence of consistent and optimal neighbor tables in nodes [8]. To implement PRR's hypercube routing scheme in a dynamic, distributed environment, such as the Internet, various protocols are needed (for node joining, leaving, table optimization, and failure recovery). In this paper, we first present a conceptual foundation, called C-set trees, *for protocol design and reasoning about consistency. We then present the detailed specification of a join protocol. In our protocol, only nodes that are joining need to keep extra state information about the join process. We present a rigorous proof that the join protocol generates consistent neighbor tables for an arbitrary number of concurrent joins. The crux of our proof is based upon induction on a C-set tree. Our join protocol can also be used for building consistent neighbor tables for a set of nodes at network initialization time. Lastly, we present both analytic and simulation results on the communication cost of a join in our protocol.*

Keywords: Peer-to-peer network, hypercube routing, overlay network, neighbor table, join protocol, consistency, C-set tree

*Research sponsored by National Science Foundation grant no. ANI-9977267 and Texas Advanced Research Program grant no. 003658–0439–2001. An abbreviated version of this report to appear in *Proceedings IEEE ICDCS*, Providence, RI, May 2003.

1. Introduction

The main goal of popular peer-to-peer systems, such as Napster [7], Gnutella [4], and Freenet [3], is object (file) sharing. Objects are stored in user machines and transferred from one machine to another upon requests. In this paper, we view such a system conceptually as a network of *nodes*. Each node, representing a user machine, can send messages to every other node in the system using the Internet.

In these systems, objects are generally replicated, with multiple copies of the same object stored in different nodes. Objects are addressed by location-independent names, with *location-independent routing* used to forward one node's query for an object to some node storing a copy of the object. Four desirable properties of a location-independent routing infrastructure for these systems, presented by Hildrum, Kubiatowicz, Rao, and Zhao in [5], are the following (slightly rephrased):

P1 Deterministic Location: If an object exists anywhere in the network, it should be located.

P2 Routing Locality: If multiple copies of an object exist in the network, a query for the object should be forwarded to a nearby copy. Also, routes should have low stretch.[1]

P3 Load Balance: The load of storing objects (or object locations) and routing information should be evenly distributed over network nodes.

P4 Dynamic Membership: The network should adapt to joining and leaving nodes while maintaining the above properties.

Napster employs a centralized directory of object locations and clearly does not satisfy P3. The system is also not fault-tolerant since the directory server is a single point of failure. Gnutella and Freenet were designed to have distributed directory information. However, neither of them satisfies P1.

In two recent research proposals, Chord [11] and CAN [9], the main operation is name resolution, i.e., mapping a name (object ID) to a node that stores a copy of the object (or the location of the object). Each system was designed to satisfy P1, P3, and P4. However, these systems do not satisfy P2 because they are not concerned with forwarding a query directly to a nearby object. Furthermore, while a name can be resolved within a small number of application-level hops,[2] the actual distance of each hop through the Internet, from one node to another, may be very large.

Of interest in this paper is the hypercube routing scheme used in PRR [8], Pastry [10], Tapestry [12], and SPRR [6]. Each node maintains a neighbor table storing pointers (IP addresses) to $O(\log n)$ nodes in the network. These tables constitute the network's routing infrastructure. With additional distributed directory information, PRR tends to satisfy each object request with a nearby copy. Given *consistent* (definition in Section 3) and *optimal* (that is, they store nearest neighbors) neighbor tables, PRR guarantees to locate an object if it exists, and the expected cost of accessing a replicated object is asymptotically optimal [8].

To implement the hypercube routing scheme in a dynamic, distributed environment, we need to address the following problems:

1. Given a set of nodes, a join protocol is needed for the nodes to initialize their neighbor tables such that the tables are *consistent*. (In what follows, a "consistent network" means a set of nodes with consistent neighbor tables.)
2. Protocols are needed for nodes to join and leave a consistent network such that the neighbor tables are still consistent after a set of joins and leaves. When a node fails, a recovery protocol is needed to re-establish consistency of neighbor tables.
3. A protocol is needed for nodes to optimize their neighbor tables.

Solving all of these problems is beyond the scope of a single paper. In this paper, we focus on designing a join protocol for the hypercube routing scheme. Given a consistent network, and a set of new nodes joining the network using our protocol, we prove that the join process will terminate and the resulting neighbor tables of both existing and new nodes are consistent (assuming reliable message delivery and no node deletion). In particular, our proof holds for *an arbitrary number of concurrent joins*.

Contributions of this paper are the following:

- We analyze the goal of the join protocol and present a conceptual foundation, called *C-set trees*, for protocol design and reasoning about consistency.
- We design a join protocol for the hypercube routing scheme, and present a detailed protocol specification. In our protocol, only nodes that are still in the join process need to keep extra state information about the join process. The join protocol can also be used for network initialization, where initially the network has only one node. Other nodes then join the network by executing the join protocol.
- We present a rigorous proof that the join protocol produces consistent neighbor tables for an arbitrary number of concurrent joins. The crux of our proof is based upon induction on a C-set tree.
- We present both analytic and simulation results on the communication cost of a join in our join protocol.

The join protocol presented in this paper provides a solution to problem 1, and part of the solution to problem 2,

[1] Stretch is the ratio between the distance traveled by a query to an object to the minimum distance between query origin and the object.

[2] The number is $O(\log n)$ for Chord and $O(dn^{1/d})$ for CAN, where n is the number of nodes in the system and d is the number of dimensions chosen for CAN.

discussed above. Moreover, the conceptual foundation presented in this paper can be used for designing protocols for leaving, failure recovery, and neighbor table optimization.[3] Note that since we are only concerned with consistency in this paper, the assumption of optimal neighbor tables is relaxed when we design our join protocol. Interested readers can refer to [5, 2] for methods of exploiting node proximity and optimizing neighbor tables.

PRR includes algorithms for dynamically maintaining directory information when objects are inserted, deleted, and accessed [8]. However it does not have node join and leave protocols. Pastry, Tapestry, and SPRR all have join and leave protocols. The issue of neighbor table consistency after concurrent joins and leaves was raised but not addressed in SPRR [6]. Pastry uses an optimistic approach to control concurrent node joins and leaves because the authors believe "contention" to be rare [10]. A join protocol was presented for Tapestry with a correctness proof [5]. The join protocol is based upon the use of multicast. The existence of a joining node is announced by a multicast message. Each intermediate node in the multicast tree keeps the joining node in a list (one list per entry updated by a joining node) until it has received acknowledgments from all downstream nodes. This approach has the disadvantage of requiring many existing nodes to store and process extra states as well as send and receive messages on behalf of joining nodes. We take a very different approach in our join protocol design. We put the burden of the join process on joining nodes only.

The balance of this paper is organized as follows: In Section 2, we briefly describe the hypercube routing scheme. In Section 3, our conceptual foundation for protocol design is illustrated. In Section 4, a detailed specification of our join protocol is presented. In Section 5, we present a consistency proof of the join protocol as well as an analysis of the protocol's communication cost. In Section 6, we discuss how to use the join protocol for network initialization, as well as several protocol enhancements. We conclude in Section 7.

2. Background: Hypercube Routing Scheme

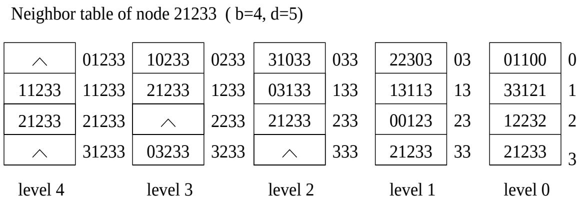
Consider a set of nodes and a set of objects. Each node or object has an identifier (ID), which is a fixed-length random binary string. (These IDs are typically generated using a hash function, such as MD5 or SHA-1.) Node and object IDs are drawn from the same ID space which can be thought of as a ring.

To present the hypercube routing scheme, we will follow notation and terminology used for PRR [8]. Each node's ID is represented by d digits of base b. For example, a 32-bit ID can be represented by 8 Hex digits ($b = 16$). Hereafter, we use $x.ID$ to denote the ID of node x and use $x[i]$, $0 \leq i \leq d-1$, to denote the ith digit in $x.ID$, with the 0th digit referred to as the *rightmost* digit.

The routing schemes of PRR [8], Pastry [10], Tapestry [12], and SPRR [6] can all be viewed as extensions of the hypercube routing scheme in this paper. For these schemes, a query of an object is routed to a node that matches the object in the largest number of suffix (or prefix) digits.[4] The schemes differ in the technique each uses to resolve the final routing hop when there are multiple nodes that match an object in the largest number of suffix (or prefix) digits. These schemes also differ in how they replicate objects and how they provide fault-tolerant routing.

2.1. Neighbor table

The neighbor table of each node consists of d levels with b entries at each level. (In what follows, we use *neighbor table* and *table* interchangeably.) The entry j at level i, $0 \leq j \leq b-1$, $0 \leq i \leq d-1$, referred to as the (i, j)-entry, in the table of node x contains link information to nodes whose IDs share the 0th to $(i-1)$th digits (that is, the rightmost i digits) with $x.ID$, and whose ith digit is j. These nodes are said to be *neighbors* of x.[5] If multiple nodes exist with the desired suffix of the (i, j)-entry, then a subset of these nodes, chosen according to some criterion, [6] may be stored in the entry with the nearest one designated as the *primary(*i, j*)-neighbor*. Each node also keeps track of its *reverse-neighbors*. Node x is a reverse(i, j)-neighbor of node y if y is the primary(i, j)-neighbor of x. Figure 1 shows an example neighbor table, in which only primary-neighbors are shown. (Also, IP addresses of neighbors are omitted.) The number to the right of each entry is the desired suffix for that entry. An empty entry indicates that there does not exist a node in the network whose ID has the desired suffix.



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

level 4		level 3		level 2		level 1		level 0	
∧	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

Figure 1. An example neighbor table

2.2. Routing scheme

When node x sends a message to node y, it first forwards the message to u_1, a primary-neighbor of x at level-0 that

[3] This paper is the first in a series of papers we write to address these problems.

[4] PRR routing uses suffix matching, while the other schemes use prefix matching.

[5] The link information for each neighbor consists of the neighbor's ID and its IP address. For simplicity, we will use "neighbor" or "node" instead of "node's ID and IP address" whenever the meaning is clear from context.

[6] In PRR, for example, nodes with minimum communication costs are chosen. Extra neighbors in an entry are used to facilitate object location [8] or for fault tolerant routing [12].

shares the rightmost digit with y. u_1 then forwards the message to its primary-neighbor at level-1 that shares the rightmost two digits with y. This process continues until the message reaches y. For example, a message sent from node 21233 to destination node 03231 is first forwarded to the primary$(0, 1)$-neighbor of 21233, which is 33121 in Figure 1, then to the primary$(1, 3)$-neighbor of 33121, say, 13331, and so on, until it reaches 03231. In this paper, the primary$(i, x[i])$-neighbor of x is chosen to be x itself. As a result, when x sends a message to y following the primary-neighbor pointers, instead of starting at level-0, it starts at level-k, where k is the length of the longest common suffix of $x.ID$ and $y.ID$.

3 Conceptual Foundation

In this section, we analyze the goals and tasks for a join protocol to produce consistent neighbor tables. We first analyze the case of a single join, which is straightforward and presents intuition of the protocol design. Then we discuss multiple joins and present the concept of *C-set trees*.

Since in this paper, we are only concerned with consistency, we relax the assumption of *optimal* neighbor tables in the hypercube routing scheme. In what follows, we use the term *neighbor* to mean *primary neighbor*. Thus, to simplify our presentation, we assume that there is only one neighbor in each table entry. Table 1 presents notation used throughout this paper.

Notation	***Definition***
$[\ell]$	the set $\{0, ..., \ell - 1\}$, ℓ is a positive integer
d	the number of digits in a node's ID
b	the base of each digit
$x.table$	the neighbor table of node x
$j \cdot \omega$	digit j concatenated with suffix ω
$N_x(i, j)$	the node in the (i, j)-entry of $x.table$, also referred to as the (i, j)-*neighbor* of node x, $i \in [b]$, $j \in [d]$
$\lvert\omega\rvert$	the number of digits in suffix ω
$csuf(\omega_1, \omega_2)$	the longest common suffix of ω_1 and ω_2
$\langle V, \mathcal{N}(V)\rangle$	a hypercube network: V is the set of nodes in the network, $\mathcal{N}(V)$ is the set of neighbor tables
$V_{l_i \ldots l_0}$	a *suffix set* of V, which is the set of nodes in V, each of which has an ID with the suffix $l_i \ldots l_0$
$\lvert V\rvert$	the number of nodes in set V

Table 1. Notation

3.1 Definitions and assumptions

Definition 3.1 *Let t_x^b be the time when node x begins joining a network, and t_x^e be the time when x becomes an S-node (to be defined in Section 4). The period from t_x^b to t_x^e, denoted by $[t_x^b, t_x^e]$, is the* **joining period** *of x.*

Definition 3.2 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, join a network. If the joining period of each node does not overlap with that of any other, then the joins are* **sequential**.

Definition 3.3 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, join a network. Let $t^b = \min(t_{x_1}^b, ..., t_{x_m}^b)$ and $t^e = \max(t_{x_1}^e, ..., t_{x_m}^e)$. If for each node x, $x \in W$, there exists a node y, $y \in W$ and $y \neq x$, such that their joining periods overlap, and there does not exist a sub-interval of $[t^b, t^e]$ that does not overlap with the joining period of any node in W, then the joins are* **concurrent**.

Definition 3.4 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 1$, join a network $\langle V, \mathcal{N}(V)\rangle$. For any node x, $x \in W$, if $V_{x[k-1]...x[0]} \neq \emptyset$ and $V_{x[k]...x[0]} = \emptyset$, $1 \leq k \leq d-1$, then $V_{x[k-1]...x[0]}$ is the* **notification set** *of x regarding V, denoted by V_x^{Notify}. If $V_{x[0]} = \emptyset$, then V_x^{Notify} is V.*

Definition 3.5 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, join a network $\langle V, \mathcal{N}(V)\rangle$. The joins are* **independent** *if for any pair of nodes x and y, $x \in W$, $y \in W$, $x \neq y$, $V_x^{Notify} \cap V_y^{Notify} = \emptyset$.*

Definition 3.6 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, join a network $\langle V, \mathcal{N}(V)\rangle$. The joins are* **dependent** *if for any pair of nodes x and y, $x \in W$, $y \in W$, $x \neq y$, one of the following is true:*

- $V_x^{Notify} \cap V_y^{Notify} \neq \emptyset$.
- *$\exists u$, $u \in W$, $u \neq x \wedge u \neq y$, such that $V_x^{Notify} \subset V_u^{Notify}$ and $V_y^{Notify} \subset V_u^{Notify}$.*

Definition 3.7 *Consider two nodes, x and y, in network $\langle V, \mathcal{N}(V)\rangle$. If there exists a neighbor sequence $(u_0, ..., u_k)$, $k \leq d$, such that u_0 is x, u_k is y, and u_{i+1} is $N_{u_i}(i, y[i])$, $i \in [k]$, then y is* **reachable** *from x (within k hops), or x can* **reach** *y, to be denoted by $\langle x \rightarrow y\rangle_k$.*

Definition 3.8 *Consider a network $\langle V, \mathcal{N}(V)\rangle$. The network, or $\mathcal{N}(V)$, is* **consistent** *if for any node x, $x \in V$, each entry in its table satisfies the following conditions:*

(a) If $V_{j \cdot x[i-1]...x[0]} \neq \emptyset$, then $N_x(i, j) = y$, $y \in V_{j \cdot x[i-1]...x[0]}$, $1 \leq i \leq d-1$, $j \in [b]$; if $V_j \neq \emptyset$, then $N_x(0, j) = y$, $y \in V_j$, $j \in [b]$.

(b) If $V_{j \cdot x[i-1]...x[0]} = \emptyset$, then $N_x(i, j) = null$, $1 \leq i \leq d-1$, $j \in [b]$; if $V_j = \emptyset$, then $N_x(0, j) = null$, $j \in [b]$.

If condition (a) in Definition 3.8 is satisfied, then $\mathcal{N}(V)$ is *false negative free*, i.e., if a node exists in the network, it is reachable from any other node. If condition (b) in Definition 3.8 is satisfied, then $\mathcal{N}(V)$ is *false positive free*, i.e., if a node does not exist in the network, then there should not exist a path that tends to lead to it.

Lemma 3.1 *In a network $\langle V, \mathcal{N}(V)\rangle$, any node is reachable from any other node iff condition (a) of Definition 3.8 is satisfied by the network.*

In designing our protocol for nodes to join a network $\langle V, \mathcal{N}(V)\rangle$, we assume that (i) V is not empty and $\mathcal{N}(V)$ is consistent, (ii) each joining node, by some means, knows a node in V initially, (iii) messages between nodes are delivered reliably, and (iv) there is no node deletion (leave or failure) during the joining period of any node.

Under the assumption that there is no node deletion during joins, condition (b) in Definition 3.8 can be satisfied easily, since once a node has joined, it always exists in the network. Hence, the goal of the join protocol is to construct neighbor tables for new nodes and update tables of existing nodes such that condition (a) in Definition 3.8 is satisfied. In a distributed peer-to-peer network, global knowledge is difficult (if not impossible) to get. Therefore, a node should utilize local information to construct or update neighbor tables. Our join protocol is designed to expand the network monotonically and preserve reachability of existing nodes so that once a set of nodes can reach each other, they always can thereafter. Hence, starting with a consistent network, $\langle V, \mathcal{N}(V)\rangle$, and a set W of joining nodes, the goals of the protocol are the following:

- **Goal 1**: $\forall x$, $\forall y$, $x \in W$, $y \in V$, eventually x and y can reach each other.
- **Goal 2**: $\forall x_1$, $\forall x_2$, $x_1 \in W$, $x_2 \in W$, eventually x_1 and x_2 can reach each other.

3.2 Operations of single join

When x joins, it is given a node g_0, $g_0 \in V$. First, x constructs its own table by copying neighbors from nodes in V level by level. It starts with copying level-0 neighbors of g_0 and filling them into level-0 of its own table. Among these level-0 neighbors, x finds node g_1, $g_1[0] = x[0]$. Then, x copies level-1 neighbors of g_1 to construct its own table at level-1, and searches for a node g_2, $g_2[1]g_2[0] = x[1]x[0]$. This process is repeated until x could not find a node g_{k+1} that shares the rightmost $k+1$ digits with it (k must exist and is at most $d-1$ since $x.ID$ is unique in the network). x then adds itself into its table. At this point, x is already able to reach any node in V.

Next, the table entries in $\mathcal{N}(V)$ that should no longer be empty because of the join of x need to be updated. Hence the nodes that have such entries need to be notified. Since there exist nodes in V that share the rightmost k digits with x but no node shares the rightmost $k+1$ digits with x, $V_{x[k-1]\ldots x[0]}$ is not empty, however, $V_{x[k]\ldots x[0]}$ is. Hence nodes in $V_{x[k-1]\ldots x[0]}$ need to be notified and their $(k, x[k])$-entries need to be updated. Conceptually, nodes in $V_{x[k-1]\ldots x[0]}$ form a forest whose roots are the level-k neighbors of x. By following neighbor pointers, x traverses the forest and notifies all nodes in $V_{x[k-1]\ldots x[0]}$ eventually. After x receives all of the replies from these nodes, it stops its join process.

During x's join, the consistency of the original network $\langle V, \mathcal{N}(V)\rangle$ is preserved because nodes in V will fill x into a table entry only if that entry is empty.

3.3 Operations of multiple joins

If multiple nodes join a network sequentially (i.e., only when the join of one node ends will another node start joining), then when a node joins, any node that joins earlier has already been integrated into the network. Hence, the joins do not interfere with each other. Also, if multiple nodes join a network concurrently and the views these nodes have about the network do not "conflict" with each other (namely, independent joins), then intuitively the joins also do not interfere with each other, because the sets of nodes these joining nodes need to notify do not intersect and none of the joining nodes needs to store any other joining node in its table. The most difficult case is *concurrent and dependent joins*, where the views different joining nodes have about the current network may conflict. For example, if node 10261 and node 00261 join concurrently, then before they know each other, each of them may think of itself as the only node with suffix 261. If handled incorrectly, the views of the joining nodes may not converge eventually, which would result in inconsistent neighbor tables.

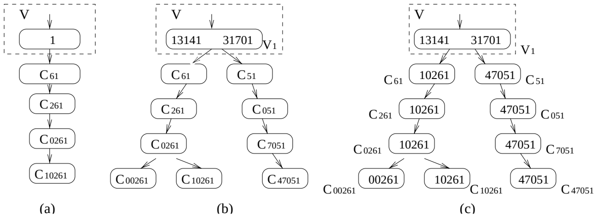


Figure 2. C-set tree

We first analyze the desirable results of multiple joins by using an example ($d = 5$, $b = 8$). Suppose a set of nodes, $W = \{10261, 47051, 00261\}$, join a consistent network $\langle V, \mathcal{N}(V)\rangle$, $V = \{72430,10353,62332,13141,31701\}$. Then, at the end of joins, for any y, $y \in V$, to reach 10261, there should exist a neighbor sequence $(u_0, u_1, \ldots, u_5)$, such that u_0 is y, u_5 is 10261, and the IDs of u_1 to u_4 have suffix 1, 61, 261, and 0261 respectively. Since $\mathcal{N}(V)$ is consistent, y must have stored a neighbor with suffix 1, which can be any node in V_1 (the set of nodes in V with suffix 1). Let the set of $(1,6)$-neighbors of nodes in V_1 be C_{61}, the set of $(2,2)$-neighbors of nodes in C_{61} be C_{261} and so on. We call these sets *C-sets* and the sequence of sets from V_1 to C_{10261} a *C-set path*. As shown in Figure 2(a), for nodes in V to reach 10261, each C-set in the path should be filled with nodes in W with the desired suffix. Similarly, for any y, $y \in V$, to reach 00261, none of the sets C_{61}, C_{261}, C_{0261} and C_{00261} should be empty. Generally, from any node in V to each node in W, there is an associated C-set path, and all the paths form a tree rooted at V_1, called a *C-set tree*, as shown in Figure 2(c). Note that C-set trees are conceptual structures used for protocol design and reasoning about consistency. They are *not implemented* in any node.

The above example is a special case of multiple joins, where the notification sets regarding V (*noti-sets*, in short) of all nodes in W are the same (namely, V_1 in the example). Generally, the noti-sets of all nodes in W may not be the same. Then, nodes in W with the same noti-set belong to the same C-set tree and the C-set trees for all nodes in W form a forest. In the above example, if $W = \{10261$, 00261, 67320, $11445\}$, then 10261 and 00261 belong to a C-set tree rooted at V_1, 67320 belongs to a C-set tree rooted at V_0 and 11445 belongs to a C-set tree rooted at V. Each C-set tree can be treated separately. In the balance of this subsection, our discussion is focused on a single C-set tree.

Definition 3.9 *Suppose a set of nodes, $W = \{x_1, ..., x_m\}$, $m \geq 1$, join a consistent network $\langle V, \mathcal{N}(V)\rangle$, and for any node x, $x \in W$, $V_x^{Notify} = V_\omega$, where $|\omega| = k$. Then the C-set* **tree template** *associated with V and W, denoted by $C(V, W)$, is defined as follows:*

- *V_ω is the root of the tree (the root is not a C-set);*
- *If $W_{l_1\cdot\omega} \neq \emptyset$, $l_1 \in [b]$, then set $C_{l_1\cdot\omega}$ is a child of V_ω, and $l_1\cdot\omega$ is the associated suffix of $C_{l_1\cdot\omega}$;*
- *If $W_{l_j\ldots l_1\cdot\omega} \neq \emptyset$, $2 \leq j \leq d-k$, l_1,...,$l_j \in [b]$, then set $C_{l_j\ldots l_1\cdot\omega}$ is a child of set $C_{l_{j-1}\ldots l_1\cdot\omega}$.*

Given V and W, the tree template is determined. Figure 2(b) shows the tree template for the above example. The task of the join protocol is to construct and update neighbor tables such that paths are established between nodes; *conceptually* nodes are filled into each C-set in $C(V, W)$. For different sequences of protocol message exchange, different nodes could be filled into each C-set, which would result in different realizations of the tree template. Figure 2(c) shows one realization of the tree template in Figure 2(b). Observe that since for any node x, $N_x(i, x[i]) = x$, $i \in [b]$, once x is filled into a C-set, it is automatically filled into those descendants of the C-set in the tree, whose suffix is also a suffix of $x.ID$. For instance, if both 13141 and 31701 store 10261 in $(1, 6)$-entry, then conceptually 10261 is filled in C_{61} and consequently, $10261 \in C_{261}$, C_{0261} and C_{10261}. C_{61} is called *the first C-set 10261 belongs to*.

Given a set W of nodes joining a consistent network $\langle V, \mathcal{N}(V)\rangle$ and nodes in W belong to the same C-set tree, we denote the C-set tree realized at the end of all joins as $cset(V, W)$ (formal definition of $cset(V, W)$ is in Section 5.1). By the end of joins, the following conditions should be satisfied by neighbor tables of nodes in $V \cup W$ for them to be consistent:

(1) $cset(V, W)$ has the same structure with $C(V, W)$ and none of the C-sets in $cset(V, W)$ is empty.

(2) For each node y, $y \in V_\omega$ (root of the C-set tree), for each child C-set of V_ω in $cset(V, W)$, y stores a node with the suffix of that C-set into its neighbor table.

(3) For each node x, $x \in W$, the C-set whose suffix is $x.ID$ is a leaf node in the tree. For any C-set along the path from this leaf node to the root, if it has any sibling C-set, then x stores a node with the suffix of that sibling C-set in its table.[7]

If condition (1) is satisfied, then in $cset(V, W)$, each leaf node whose suffix corresponds to a node's ID must include that node. Therefore, the union of all C-sets in $cset(V, W)$ is W. If condition (2) is satisfied, then all entries in $\mathcal{N}(V)$ that need to be updated are updated. If condition (3) is satisfied, then for any node in W, it can reach every other node in W. Hence, these three conditions, together with each joining node's copying neighbors from nodes in V, ensure that the network is consistent after the joins.

4 Specification of Join Protocol

In our protocol, each node keeps its own status, which could be *copying*, *waiting*, *notifying*, and *in_system*. When a node starts joining, its status is set to *copying*. Each node also stores the state of each neighbor as T or S in its table, where S indicates that the neighbor is in status *in_system*, while T means it is not yet.

A node with status *in_system* is called an *S-node*; otherwise, it is a *T-node*. The phase in which a node, x, is in status *copying*, *waiting* or *notifying* is called c-phase, w-phase or n-phase, respectively. In c-phase, x copies neighbors and constructs most part of its table. In w-phase, x waits until it is filled in the table of an S-node, which indicates that conceptually, x is filled into a C-set of a C-set tree. In n-phase, x seeks and notifies nodes that are in the subtree rooted at the parent C-set of the one x is filled into. When x finds no more node to notify, it changes status to *in_system* and becomes an S-node. Figure 3 describes the state variables of a joining node. Variables in the first part are also used by nodes in V, where for each node y, $y \in V$, $y.status =$ *in_system*, $y.table$ is populated in a way that satisfies the conditions in Definition 3.8, and $N_y(i, j).state = S$ if $N_y(i, j) \neq$ *null* for all i and j. Figure 4 presents the join protocol messages. Figure 5 to Figure 14 present the pseudo-code of the protocol, in which x, y, u and v denote nodes, and i, j and k denote integers.

When any node, x, sets $N_x(i, j) = y$, $y \neq x$, x needs to sends a *RvNghNotiMsg(y,*$N_x(i, j).state$*)* to y, and y should reply to x if $N_x(i, j).state$ is not consistent with $y.status$. For clarity of presentation, we have omitted the sending and reception of these messages in the pseudo-code.

4.1 Action in status *copying*

In c-phase, x constructs its table level by level until it stops at level-k, $k \in [b]$, where after copying level-k neighbors of node g_k, x could not find a node g_{k+1} that shares the

[7] For instance, in Figure 2(c), from C_{00261} to V_1, there are two branches with suffix 10261 and 51 respectively. Hence, by the end of joins, 00261 should have stored a node with suffix 10261 and a node with suffix 51 in its table.

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*.

Figure 3. State variables

Messages exchanged by nodes:

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 $s = T$.

Figure 4. Protocol messages

rightmost $k+1$ digits with it, or x finds such a g_{k+1} but g_{k+1} is still a *T-node*. In the former case, x sends a *JoinWaitMsg* to g_k, while in the latter case, x sends a *JoinWaitMsg* to g_{k+1}. Meanwhile, x sets its status to *waiting*. Figure 5 depicts the action in c-phase. (For clarity of presentation, we have omitted the sending of a *CpRstMsg* from x to g, and the reception of a *CpRlyMsg* from g to x.)

4.2 Action in status *waiting*

The *JoinWaitMsg* x sends to g_k (or g_{k+1}) serves as a notification to g_k (or g_{k+1}) that x is waiting to be stored in its table. If g_k (or g_{k+1}) has already stored node u_1 in the table entry x can be filled into by the time it receives the message, it sends a negative reply to x with u_1 and its own table. x then sends another *JoinWaitMsg* to u_1. This process may be repeated (for at most d times) until some node fills x into its table and sends x a positive reply. Note that a node can only reply to x when it is an S-node; otherwise, it has to delay its reply. On receiving a positive reply, x changes status to *notifying*. Meanwhile, x sets $x.noti\_level$ to $|csuf(x.ID, y.ID)|$, where y is the node that sends the positive reply to x. Figure 6 and Figure 7 present actions

```
Action of $x$ on joining $\langle V, \mathcal{N}(V)\rangle$, given node $g_0$, $g_0 \in V$:

$i$: an integer, initially 0. $p$, $g$: a node, initially $g_0$.
$s \in \{T, S\}$, initially $S$. // $s$ records the status of node $g$

$x.status$ = copying;
while ($g \neq$ null and $s$ == $S$) { // copy level-$i$ neighbors of $g$
  for ($j$ = 0; $j < b$; $j$++) {
    $N_x(i,j)$ = $N_g(i,j)$; $N_x(i,j).state$ = $N_g(i,j).state$;
  }
  $p$ = $g$; $g$ = $N_p(i, x[i])$; $s$ = $N_p(i, x[i]).state$; $i$++;
}
for ($i$ = 0; $i < d$; $i$++) {$N_x(i, x[i])$ = $x$; $N_x(i, x[i]).state$ = $T$;}
$x.status$ = waiting;
if ($g$ == null) {
  Send JoinWaitMsg to $p$; $Q_n$ = $Q_n \cup \{p\}$; $Q_r$ = $Q_r \cup \{p\}$;
}else{ // $g \neq$null and $s$ == $T$
  Send JoinWaitMsg to $g$; $Q_n$ = $Q_n \cup \{g\}$; $Q_r$ = $Q_r \cup \{g\}$;
}
```

Figure 5. Action in status copying

upon receiving *JoinWaitMsg* and *JoinWaitRlyMsg*, respectively.

```
Action of $y$ on receiving JoinWaitMsg from $x$:

$k$ = $|csuf(x.ID, y.ID)|$;
if ($y.status$ == in_system) {
  if ($N_y(k, x[k]) \neq$ null $\wedge$ $N_y(k, x[k]) \neq x$ ) {
    Send JoinWaitRlyMsg(negative, $N_y(k, x[k])$, $y.table$) to $x$;
  } else { // it must be that $N_y(k, x[k])$ is null
    $N_y(k, x[k])$ = $x$; $N_y(k, x[k]).state$ = $T$;
    Send JoinWaitRlyMsg(positive, $x$, $y.table$) to $x$;
  }
}else $Q_j$ = $Q_j \cup \{x\}$;
```

Figure 6. Action on receiving JoinWaitMsg

```
Action of $x$ on receiving JoinWaitRlyMsg($r$, $u$, $y.table$) from $y$:

$Q_r$ = $Q_r - \{y\}$; $k$ = $|csuf(x.ID, y.ID)|$;
if ($N_x(k, y[k])$ == $y$) $N_x(k, y[k]).state$ = $S$;
if ($r$ == positive) {
  $x.status$ = notifying; $x.noti\_level$ = $k$;
  $R_x(k, x[k])$ = $R_x(k, x[k]) \cup \{y\}$;
}else{ // a negative reply, needs to send a JoinWaitMsg to $u$
  Send JoinWaitMsg to $u$; $Q_n$ = $Q_n \cup \{u\}$; $Q_r$ = $Q_r \cup \{u\}$;
}
Check_Ngh_Table($y.table$);
if ($x.status$ == notifying $\wedge$ $Q_r$ == $\phi$ $\wedge$ $Q_{sr}$ == $\phi$)
  Switch_To_S_Node();
```

Figure 7. Action on receiving JoinWaitRlyMsg

4.3 Action in status *notifying*

As shown in Figure 8, in n-phase, if x finds a node y such that $|csuf(x.ID, y.ID)| \geq x.noti\_level$, x sends a *JoinNotiMsg* to y if it has not done so. Note that when x sends out a *JoinNotiMsg*, it needs to include $x.table$ in the message. On receiving the *JoinNotiMsg* from x, y fills x into its table if the corresponding entry is empty and sends a *JoinNotiRlyMsg* that includes $y.table$ to x. x then checks

$y.table$ level by level to send more *JoinNotiMsg* if necessary. Also, if x finds a node u in $y.table$ that can be filled into an empty entry, it stores u in that entry. Figure 9 and Figure 10 present the actions on receiving *JoinNotiMsg* and *JoinNotiRlyMsg*, respectively.

```
Check_Ngh_Table($y.table$) at $x$:

for ($i$ = 0; $i < d$, $i$++) {
  for ($j$ = 0; $j < b$; $j$++) {
    if ($N_y(i,j) \neq$ null$\wedge$ $N_y(i,j) \neq x$){
      $u$ = $N_y(i,j)$; $k$ = $|csuf(x.ID, u.ID)|$;
      if ($N_x(k, u[k])$==null) {
        $N_x(k, u[k])$ = $u$; $N_x(k, u[k]).state$ = $N_y(i,j).state$;
      }
      if ($x.status$ == notifying $\wedge$ $k \geq x.noti\_level$ $\wedge$ $u \notin Q_n$){
        Send JoinNotiMsg($x.table$) to $u$;
        $Q_n$ = $Q_n \cup \{u\}$; $Q_r$ = $Q_r \cup \{u\}$;
      }
    }
  }
}
```

Figure 8. Subroutine: Check_Ngh_Table

```
Action of $y$ on receiving JoinNotiMsg($x.table$) from $x$:

$k$ = $|csuf(x.ID, y.ID)|$; $f$ = false;
if ($N_y(k, x[k])$ == null) {
  $N_y(k, x[k])$ = $x$; $N_y(k, x[k]).state$ = $T$;
}
if ($N_x(k, y[k]) \neq y$ $\wedge$ $y.status$ == in_system) $f$ = true;
if ($N_y(k, x[k])$ == $x$)
  Send JoinNotiRlyMsg(positive, $y.table$, $f$) to $x$;
else Send JoinNotiRlyMsg(negative, $y.table$, $f$) to $x$;
Check_Ngh_Table($x.table$);
```

Figure 9. Action on receiving JoinNotiMsg

```
Action of $x$ on receiving JoinNotiRlyMsg($r$, $y.table$, $f$) from $y$:

$Q_r$ = $Q_r - \{y\}$; $k$ = $|csuf(x.ID, y.ID)|$;
if ($r$ == positive) $R_x(k, x[k])$ = $R_x(k, x[k]) \cup \{y\}$;
if ($f$ == true $\wedge$ $k > x.noti\_level$ $\wedge$ $y \notin Q_{sn}$){
  Send SpeNotiMsg($x$,$y$) to $N_x(k, y[k])$;
  $Q_{sn}$ = $Q_{sn} \cup \{y\}$; $Q_{sr}$ = $Q_{sr} \cup \{y\}$;
}
Check_Ngh_Table($y.table$);
if ($Q_r$ == $\phi$ $\wedge$ $Q_{sr}$ == $\phi$) Switch_To_S_Node();
```

Figure 10. Action on receiving JoinNotiRlyMsg

In what follows, we use "notification" to refer to either a *JoinWaitMsg* or a *JoinNotiMsg*. So far, three cases for a node x to know any other node, y, have been presented: (i) x copies y during c-phase, (ii) x receives a notification from y, and (iii) x receives a message from z, which includes $z.table$, and y is in $z.table$. There is one more case, as shown in Figures 9 and 10: Suppose in status *notifying*, x sends a *JoinNotiMsg* to y. When y receives the message, if y is an S-node and finds that $N_x(k, y[k]) = u_1$, where $k = |csuf(x.ID, y.ID)|$ and $u_1 \neq y$, then y sets a flag in its reply. Seeing the flag in the reply, x sends a *SpeNotiMsg* to u_1 to inform it about y if x has not done so and $k > x.noti\_level$. If u_1 has set u_2 instead of y as the corresponding neighbor, it forwards the message to u_2. This process stops when an informed node stores or has stored y in its table and sends a reply to x. (This process can be repeated at most d times.) Figure 11 and Figure 12 depict the actions on receiving *SpeNotiMsg* and *SpeNotiRlyMsg*, respectively.

```
Action of $u$ on receiving SpeNotiMsg($x$, $y$) from $v$:

$k$ = $|csuf(y.ID, u.ID)|$;
if ($N_u(k, y[k])$==null) {
  $N_u(k, y[k])$ = $y$; $N_u(k, y[k]).state$ = $S$;
}
if ($N_u(k, y[k]) \neq y$) Send SpeNotiMsg($x$, $y$) to $N_u(k, y[k])$;
else Send SpeNotiRlyMsg($x$, $y$) to $x$;
```

Figure 11. Action on receiving SpeNotiMsg

```
Action of $x$ on receiving SpeNotiRlyMsg($x$, $y$) from $u$:

$Q_{sr}$ = $Q_{sr} - \{y\}$; if ($Q_r$==$\phi$ $\wedge$ $Q_{sr}$==$\phi$) Switch_To_S_Node();
```

Figure 12. Action on receiving SpeNotiRlyMsg

4.4 Action in status *in_system*

When x has received replies from all of the nodes it has notified and finds no more node to notify, it changes status to *in_system*. Next, x informs all of its reverse-neighbors and nodes in Q_j, which have sent it a *JoinWaitMsg*, that it has become an *S-node*. Figure 13 and Figure 14 present the pseudo-code for this part.

```
Switch_To_S_Node() at $x$:

$x.status$ = in_system;
for ($i$ = 0; $i < d$; $i$++) { $N_x(i, x[i]).state$ = $S$; }
for each $v$ of $x$'s reverse neighbors, Send InSysNotiMsg to $v$;
for each $u$, $u \in Q_j$ {
  $k$ = $|csuf(x.ID, u.ID)|$;
  if ($N_x(k, u[k])$==null) {
    $N_x(k, u[k])$ = $u$; $N_x(k, u[k]).state$ = $T$;
    Send JoinWaitRlyMsg(positive, $u$, $x.table$) to $u$;
  }else Send JoinWaitRlyMsg(negative,$N_x(k, u[k])$,$x.table$) to $u$;
}
```

Figure 13. Subroutine: Switch_To_S_Node

5 Protocol Analysis

In this section, we present a consistency proof of the join protocol, and analyze the communication cost of each join. Here we only present important lemmas and proof outlines. Proof details can be found in Appendix A.

```
Action of $y$ on receiving a InSysNotiMsg from $x$:
  $k = |csuf(y.ID, x.ID)|$; $N_y(k, x[k]).state = S$;
```

Figure 14. Action on receiving InSysNotiMsg

5.1 Correctness of join protocol

We present two theorems. Theorem 1 states that when a set of nodes use the join protocol to join a consistent network, then at the end of the joins, the resulting network is also consistent. Theorem 2 states that each joining node eventually becomes an S-node. We begin by presenting Lemmas 5.1 to 5.5. Recall that t^e_x denotes the time when a joining node x becomes an S-node. In what follows, we use t^e to denote $\max(t^e_{x_1}, ..., t^e_{x_m})$.

Lemma 5.1 *Suppose node x joins a consistent network $\langle V, \mathcal{N}(V)\rangle$. Then, at time t^e_x, $\langle V \cup \{x\}, \mathcal{N}(V \cup \{x\})\rangle$ is consistent.*

Lemma 5.2 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, join a consistent network $\langle V, \mathcal{N}(V)\rangle$ sequentially. Then, at time t^e, $\langle V \cup W, \mathcal{N}(V \cup W)\rangle$ is consistent.*

Lemma 5.3 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, join a consistent network $\langle V, \mathcal{N}(V)\rangle$ concurrently. If the joins are independent, then at time t^e, $\langle V \cup W, \mathcal{N}(V \cup W)\rangle$ is consistent.*

Lemma 5.4 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, join a consistent network $\langle V, \mathcal{N}(V)\rangle$ concurrently. If the joins are dependent, then at time t^e, $\langle V \cup W, \mathcal{N}(V \cup W)\rangle$ is consistent.*

To prove Lemma 5.4, first consider any two nodes in W, x and y. If $V^{Notify}_x = V^{Notify}_y$, then x and y belong to the same C-set tree rooted at V^{Notify}_x, otherwise they belong to different C-set trees. We consider nodes in the same C-set tree first. We next present the definition of $cset(V, W)$, the C-set tree realized at time t^e. The definition is based on a snapshot of neighbor tables at time t^e. Then we prove that the three conditions stated in Section 3.3 are satisfied by $cset(V, W)$ and neighbor tables of nodes in $V \cup W$.

Definition 5.1 *Suppose a set of nodes, $W = \{x_1, ..., x_m\}$, $m \geq 1$, join a consistent network $\langle V, \mathcal{N}(V)\rangle$, and for any node x, $x \in W$, $V^{Notify}_x = V_\omega$, $|\omega| = k$. Then the C-set tree realized at time t^e, denoted as $cset(V, W)$, is defined as follows:*

- *V_ω is the root of the tree.*
- *$C_{l_1 \cdot \omega}$ is a child of V_ω, where $C_{l_1\cdot\omega} = \{x, x \in W_{l_1\cdot\omega} \wedge (\exists u, u \in V_\omega \wedge N_u(k, l_1) = x)\}$, $l_1 \in [b]$.*
- *$C_{l_j...l_1\cdot\omega}$ is a child of $C_{l_{j-1}...l_1\cdot\omega}$, where $C_{l_j...l_1\cdot\omega} = \{x, x \in W_{l_j...l_1\cdot\omega} \wedge (\exists u, u \in C_{l_{j-1}...l_1\cdot\omega} \wedge N_u(k + j - 1, l_j) = x)\}$, $2 \leq j < d - k$, $l_1,...,l_j \in [b]$.*

Intuitively, in $cset(V, W)$, $C_{l_1\cdot\omega}$ is the set of nodes in $W_{l_1\cdot\omega}$, each of which is stored as a (k, l_1)-neighbor by at least one node in V_ω by time t^e; $C_{l_2 l_1\cdot\omega}$ is the set of nodes in $W_{l_2 l_1\cdot\omega}$, each of which is stored as a $(k + 1, l_2)$-neighbor by at least one node in $C_{l_1\cdot\omega}$ by time t^e, and so on. Next, we prove a few propositions about $cset(V, W)$ and $\mathcal{N}(V \cup W)$, given that nodes in W belong to the same C-set tree. Propositions 5.1, 5.2 and 5.3 state that condition (1), (2) and (3), stated in Section 3.3, are satisfied at time t^e, respectively, while Proposition 5.4 concludes that by time t^e, nodes in the same C-set tree as well as nodes in V can reach each other. Then, Proposition 5.5 extends the result to nodes in different C-set trees. Our proofs of these propositions are based upon induction on C-set trees. With these propositions, we can prove Lemma 5.4, Lemma 5.5 and finally prove Theorem 1. Note that Propositions 5.1 to 5.4 make the following assumption:

Assumption 5.1 *(for Propositions 5.1 to 5.4)*
A set of nodes, $W = \{x_1, ..., x_m\}$, $m \geq 2$, join a consistent network $\langle V, \mathcal{N}(V)\rangle$ concurrently and for any x, $x \in W$, $V^{Notify}_x = V_\omega$, $|\omega| = k$.

Proposition 5.1 *If $W_{l_j...l_1\cdot\omega} \neq \emptyset$, $1 \leq j \leq d - k$, $l_1,...,l_j \in [b]$, then $C_{l_j...l_1\cdot\omega} \neq \emptyset$.*

Proposition 5.2 *Let u be a node in V_ω. If $W_{l_1\cdot\omega} \neq \emptyset$, $l_1 \in [b]$, then there exists a node x, $x \in W_{l_1\cdot\omega}$, such that $N_u(k, l_1) = x$ by time t^e.*

Proposition 5.3 *For any node x, $x \in W$, if $W_{l\cdot l_i...l_1\cdot\omega} \neq \emptyset$, where $l \in [b]$ and $l_i...l_1\cdot\omega$ is a suffix of $x.ID$, $1 \leq i < d - k$, then $N_x(i + k, l) = y$ by time t^e, $y \in W_{l\cdot l_i...l_1\cdot\omega}$; if $W_{l\cdot\omega} \neq \emptyset$, $l \in [b]$, then $N_x(k, l) = y$, $y \in W_{l\cdot\omega}$.*

Proposition 5.4 *For any two nodes x and y, $x \in V \cup W$, $y \in V \cup W$, $\langle x \to y\rangle_d$ by time t^e.*

Proposition 5.5 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, join a consistent network $\langle V, \mathcal{N}(V)\rangle$ concurrently. Let $G(V_{\omega_1}) = \{x, x \in W, V^{Notify}_x = V_{\omega_1}\}$, $G(V_{\omega_2}) = \{y, y \in W, V^{Notify}_y = V_{\omega_2}\}$, $\omega_1 \neq \omega_2$. Then by time t^e,*

- *$\forall x, \forall y$, $x \in G(V_{\omega_1})$, $y \in G(V_{\omega_2})$, $\langle x \to y\rangle_d$.*

Proof of Lemma 5.4: First, separate nodes in W into groups $\{G(V_{\omega_i}), 1 \leq i \leq h\}$, where $\omega_i \neq \omega_j$ if $i \neq j$, such that for any node x in W, $x \in G(V_{\omega_i})$ iff $V^{Notify}_x = V_{\omega_i}$, $1 \leq i \leq h$. Then, by Propositions 5.4, 5.5, and Lemma 3.1, the lemma follows. ■

Lemma 5.5 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, join a consistent network $\langle V, \mathcal{N}(V)\rangle$ concurrently. Then at time t^e, $\langle V \cup W, \mathcal{N}(V \cup W)\rangle$ is consistent.*

Proof of Lemma 5.5: First, separate nodes in W into groups, such that joins of nodes in the same group are dependent and joins of nodes in different groups are mutually independent, as follows (initially, let $i = 1$ and put an arbitrary node x, $x \in W$, in G_1):

- For each node y, $y \in W - \bigcup_{j=1}^{i} G_j$, if there exists a node x, $x \in G_i$, such that $(V_y^{Notify} \cap V_x^{Notify} \neq \emptyset)$ or $(\exists u, u \in W - \bigcup_{j=1}^{i-1} G_j$, $(V_y^{Notify} \subset V_u^{Notify}) \wedge (V_x^{Notify} \subset V_u^{Notify}))$, put y in G_i;
- Pick any node x', $x' \in W - \bigcup_{j=1}^{i} G_j$, put x' in G_{i+1}, increment i and repeat these two steps until there is no node left.

Then, by Lemmas 5.4 and 5.3, the lemma holds. ■

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.*

Proof of Theorem 1: According to their joining periods, nodes in W can be separated into several groups, $\{G_i$, $1 \leq i \leq l\}$, such that nodes in the same group join concurrently and nodes in different groups join sequentially. Let the joining period of G_i be $[t_{G_i}^b, t_{G_i}^e]$, $1 \leq i \leq l$, where $t_{G_i}^b = \min(t_x^b, x \in G_i)$ and $t_{G_i}^e = \max(t_x^e, x \in G_i)$. We number the groups in such a way that $t_{G_i}^e \leq t_{G_{i+1}}^b$. Then, by Lemma 5.1 and Lemma 5.5, we conclude that at time t^e, $\langle V \cup W, \mathcal{N}(V \cup W)\rangle$ is consistent. ■

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.*

We present a proof outline here. Our proof is based upon the assumption of reliable message delivery and no node deletion during joins. First, consider a joining node, x, in c-phase. x eventually enters w-phase because it sends at most d *CpRstMsg*. Each receiver of a *CpRstMsg* replies to x with no waiting.

Second, consider a joining node, x, in w-phase. In this phase, x sends *JoinWaitMsg* to at most d nodes. We next show that after sending a *JoinWaitMsg*, x eventually receives a reply. If the receiver of a *JoinWaitMsg*, y, is an S-node, then y replies with no waiting. If y is not yet an S-node, then it is a joining node in n-phase and will wait until it becomes an S-node before replying to x. Thus, to complete the proof of this theorem, it suffices to show that any joining node in n-phase eventually becomes an S-node.

Last, consider a joining node, z, in n-phase. There are two types of messages sent by z in this phase, *JoinNotiMsg* and *SpeNotiMsg*. z only sends *JoinNotiMsg* to a subset of nodes in $V \cup W$ that share the rightmost i digits with itself, $i = z.noti\_level$, and each receiver of a *JoinNotiMsg* replies to z with no waiting. Also, z only sends *SpeNotiMsg* to a subset of nodes in W that share the rightmost $i + 1$ digits with it.[8] Each *SpeNotiMsg* is forwarded at most d times before a reply is sent to z, and each receiver of the message can reply to z or forward the message to another node with no waiting. Therefore, z eventually becomes an S-node.

5.2. Communication cost

Among the messages exchanged during a node's join, *CpRstMsg*, *JoinWaitMsg*, *JoinNotiMsg*, and their corresponding replies could be big in size since a copy of a neighbor table may be included, while messages of other types (*InSysNotiMsg*, *SpeNotiMsg*, *SpeNotiRlyMsg*, *RvNghNotiMsg*, and *RvNghNotiRlyMsg*) are small in size. We analyze the number of big messages in this section. The analyses for numbers of small messages are presented in Appendix A.2.

For each message of type *CpRstMsg*, *JoinWaitMsg*, or *JoinNotiMsg*, there is one and only one corresponding reply. Hence, it is sufficient to analyze the number of messages for these three types. Theorem 3 presents an upper bound of the total number of *CpRstMsg* and *JoinWaitMsg* sent by a joining node, x. Next, let J be the number of *JoinNotiMsg* sent by x. The expectation of J when only x joins is given by Theorem 4, and an upper bound of the expectation of J when x joins concurrently with other nodes is given by Theorem 5.

Theorem 3 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 1$, join a consistent network $\langle V, \mathcal{N}(V)\rangle$. Then, for any x, $x \in W$, the number of* CpRstMsg *and* JoinWaitMsg *sent by x is at most $d + 1$.*

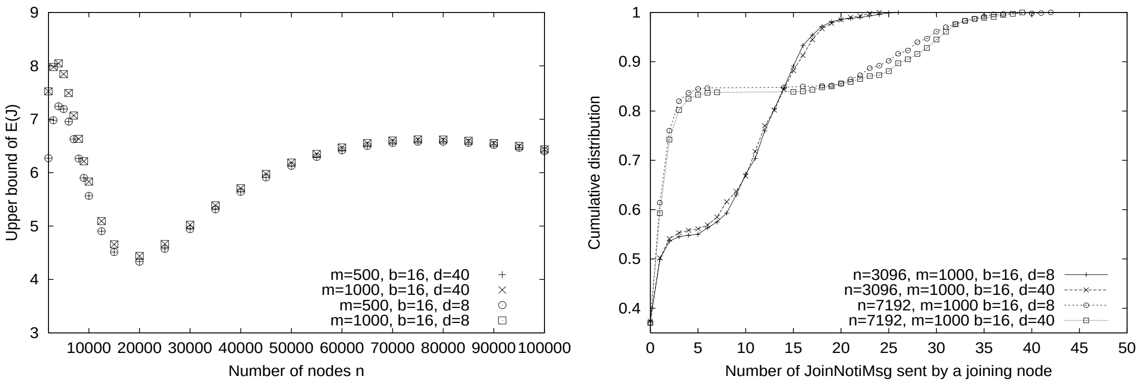
Theorem 4 *Suppose node x joins a consistent network $\langle V, \mathcal{N}(V)\rangle$, $|V| = n$. Then, the expected number of JoinNotiMsg sent by x is $\sum_{i=0}^{d-1} \frac{n}{b^i} P_i(n) - 1$, where $P_i(n)$ is $\sum_{k=1}^{\min(n,B)} \frac{C(B,k)C(b^d - b^{d-i}, n-k)}{C(b^d - 1, n)}$ for $1 \leq i < d - 1$, where $B = (b-1)b^{d-1-i}$ and $C(B, k)$ denotes number of k-combinations of B objects, $P_0(n)$ is $\frac{C(b^d - b^{d-1}, n)}{C(b^d - 1, n)}$, and $P_{d-1}(n)$ is $1 - \sum_{j=0}^{d-2} P_j(n)$.*

Theorem 5 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, join a consistent network $\langle V, \mathcal{N}(V)\rangle$. Then for any node x, $x \in W$, an upper bound of the expected number of JoinNotiMsg sent by x is $\sum_{i=0}^{d-1} (\frac{n+m}{b^i}) P_i(n)$, where $n = |V|$, and $P_i(n)$ is defined in Theorem 4.*

Proofs of the above theorems are presented in Appendix A.2. Here we only present the intuition for proving Theorem 4. Suppose $V_x^{Notify} = V_\omega$. Since only x joins, x needs to send *JoinNotiMsg* to all nodes in V_ω, except the one it sends *JoinWaitMsg* to. Let $Z = |V_\omega|$ and $Y = |\omega|$. Hence, $E(J) = E(Z - 1)$, where $E(Z) = E(E(Z|Y)) = \sum_{i=0}^{d-1} (E(Z|Y = i)) P_Y(Y = i)$. It can then be proved that $E(Z|Y = i) = \frac{n}{b^i}$ and $P_Y(Y = i) = P_i(n)$.

Figure 15(a) plots the upper bound of $E(J)$ when a set of nodes join concurrently, where $n = |V|$ and $m = |W|$. We have implemented our join protocol in detail in an event-driven simulator. Figure 15(b) shows simulation results of the number of *JoinNotiMsg* sent by each joining node. We

[8] In simulations, we observed that a joining node rarely sends a *SpeNotiMsg*.



(a) Theoretical upper bound

(b) Data from simulations

Figure 15. Number of JoinNotiMsg sent by a joining node

use the GT-ITM package [1] to generate network topologies. The topology used in Figure 15(b) has 8320 routers. There are two simulation setups. In one setup, 4096 nodes (end-hosts) are attached to the routers randomly, 3096 of which form a consistent network initially and the remaining 1000 nodes join concurrently. In the other setup, 8192 nodes are attached, and 1000 nodes join a consistent network formed by the other 7192 nodes. (In the simulations, all joins start at the same time.) For the simulations shown in Figure 15(b), average number of *JoinNotiMsg* sent by joining nodes are 6.117, 6.051, 5.026, and 5.399, respectively, while the upper bounds by Theorem 5 are 8.001, 8.001, 6.986, and 6.986, respectively. Also, results in Figure 15(b) indicate that the majority of joining nodes send a small number of *JoinNotiMsg*. Other simulation results show the same trend.

6 Discussions

6.1 Network initialization

The join protocol can be used for network initialization. To initialize a network with n nodes, put one node, x, in V, and construct $x.table$ 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 $j \neq x[i]$.

Next, the other $n-1$ nodes join the network by executing the join protocol, each is given x to begin with. Eventually, a consistent network is constructed.

6.2 Message size reduction

In the join protocol, some types of messages need to include a copy of the sender's neighbor table. Several enhancements can be made to reduce the size of such a message:

- When node x sends a *JoinNotiMsg* to node y, it does not need to include its whole table in the message. Only including level-i, $i = x.noti\_level$, to level-k, $k = |csuf(x.ID, y.ID)|$, is enough.
- Moreover, x can include a *bit vector* in the *JoinNotiMsg* it sends to node y, as suggested in [5]. Each bit corresponds to an entry in $x.table$, with '1' meaning that the entry is already filled and '0' meaning the opposite. Then, in its reply to x, y only needs to include neighbors in level-i entries that correspond to a '0' in the bit vector, $0 \leq i < x.noti\_level$, as well as all level-i' neighbors, $x.noti\_level \leq i' \leq d-1$.

6.3 Neighbor table optimization

There are several ways to optimize a node's neighbor table. One is to copy neighbors from nearby nodes. For example, instead of copying level-0 neighbors of g_0, node x can choose a node that is closest to it from among g_0 and neighbors of g_0 (or even neighbors of neighbors of g_0), and copy level-0 neighbors of that node to construct its own table at level-0. As suggested by [2], copying neighbors from nearby nodes help exploit node proximity in the underlying network. x can also optimize its table after its join process by running some algorithm (for example, the nearest neighbor algorithm presented in [5]) to locate nearest neighbors. In presenting our join protocol, we previously assumed that once a table entry is filled, it will not be modified thereafter. This assumption was made to ensure that reachability of each node is preserved. If neighbor tables can be optimized without sacrificing the reachability of any node, then consistency of neighbor tables will not be affected by neighbor replacements.

7. Conclusions

For the hypercube routing scheme used in several proposed peer-to-peer systems [8, 12, 10, 6], we present a new join protocol that constructs neighbor tables for new nodes and updates neighbor tables in existing nodes. We present a rigorous proof that the join protocol produces consistent neighbor tables after an arbitrary number of concurrent joins. Furthermore, we present a conceptual foundation, C-set trees, for reasoning about consistency. We plan to use this conceptual foundation to design protocols for leaving, failure recovery, and neighbor table optimization. The expected communication cost of integrating a new node into the network is shown to be small by both theoretical analysis and simulations.

Acknowledgment

The authors would like to thank Greg Plaxton for valuable discussions.

References

[1] K. Calvert, M. Doar, and E. W. Zegura. Modeling Internet Topology. *IEEE Communications Magazine*, June 1997.

[2] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Exploiting network proximity in peer-to-peer overlay networks. In *Proc. of International Workshop on Future Directions in Distributed Computing*, 2002.

[3] Freenet. http://freenetproject.org.

[4] Gnutella. http://www.gnutella.com.

[5] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao. Distributed object location in a dynamic network. In *Proc. of ACM Symposium on Parallel Algorithms and Architectures*, 2002.

[6] X. Li and C. G. Plaxton. On name resolution in peer-to-peer networks. In *Proc. of the 2nd Workshop on Principles of Mobile Computing*, 2002.

[7] Napster. http://www.napster.com/.

[8] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. of ACM Symposium on Parallel Algorithms and Architectures*, 1997.

[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of ACM SIGCOMM*, 2001.

[10] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of IFIP/ACM International Conference on Distributed Systems Platforms*, 2001.

[11] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of ACM SIGCOMM*, 2001.

[12] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, Aug. 2001.

A Proofs of Propositions, Lemmas and Theorems

A.1 Correctness of Join Protocol

In this section, we present our proofs of propositions, lemmas and theorems stated in Section 5. The notation used in our proofs is shown in Table 2. Also, in what follows, when we mention that *node y stops at level-j at the end of c-phase*, $j \in [b]$, we mean that (i) after copying level-j neighbors of u_j, y could not find a node that shares the rightmost $j+1$ digits with it, or (ii) after copying level-$(j-1)$ neighbors from u_{j-1}, y finds that u_j shares the rightmost j digits with it, but u_j is still a T-node. Thirdly, recall that notation $\langle x \rightarrow y \rangle_d$, introduced in Section 3, denotes that x can reach y within d hops. By the definition, $\langle x \rightarrow y \rangle_d$ indicates that there exists a neighbor sequence $(u_0, ..., u_d)$ such that u_0 is x, u_d is y, and u_{i+1} is $N_{u_i}(i, y[i])$, $i \in [d]$. Observe that $u_j = x$, $0 \leq j \leq k$, $k = |csuf(x.ID, y.ID)|$. Hence, it is sufficient to say that if $\langle x \rightarrow y \rangle_d$, then there exists a neighbor sequence $(u_k, ..., u_d)$, $k = |csuf(x.ID, y.ID)|$, such that $u_k = x$, $u_d = y$, and u_{i+1} is $N_{u_i}(i, y[i])$ for $k \leq i \leq d-1$.

Again, we use "notification" to refer to a *JoinWaitMsg* or a *JoinNotiMsg*. We also use the following abbreviations for protocol messages:

CP	for	*CpRstMsg*
CPRly	for	*CpRlyMsg*
JW	for	*JoinWaitMsg*
JWRly	for	*JoinWaitRlyMsg*
JN	for	*JoinNotiMsg*
JNRly	for	*JoinNotiRlyMsg*
SN	for	*SpeNotiMsg*
SNRly	for	*SpeNotiRlyMsg*
RN	for	*RvNghNotiMsg*
RNRly	for	*RvNghNotiRlyMsg*

Notation	*Definition*
$x \overset{j}{\rightarrow} y$	the action that x sends a *JN* or a *JW* to y
$x \overset{jn}{\rightarrow} y$	the action that x sends a *JN* to y
$x \overset{jw}{\rightarrow} y$	the action that x sends a *JW* to y
$x \overset{c}{\rightarrow} y$	the action that x sends a *CP* to y
$x[k-1]...x[0]$	the rightmost k digits of $x.ID$; if $k = 0$, then it denotes the empty string
$A(x)$	the **attaching-node** of x, which is the node that sends a positive *JWRly* to x
t_x^e	the time x changes status to *in_system*, i.e., the end of x's join process,
t^e	$\max(t_{x_1}^e, ..., t_{x_m}^e)$

Table 2. Notation in proofs

In our proofs, we assume reliable message delivery and no node deletion during the joins. We also assume that the actions specified by Figures 6, 7, 9, 10, 11, 12 and 14 are atomic. The following facts, which are easily observed from the join protocol, are used frequently in the proofs.

Fact 1 *Messages of type* CP, JW, *and* JN *are only sent by T-nodes.*

Fact 2 *If node x sends out a* JWRly *at time t, then x is already an S-node at time t.*

Fact 3 *If $A(x) = u$, then $x.noti\_level = h$ and $N_u(h, x[h]) = x$, where $h = |csuf(x.ID, u.ID)|$. Also, x changes status from waiting to notifying immediately after it receives a* JWRly *from u.*

Fact 4 *A joining node, x, only sends a* JN *to y if x is in status notifying and $|csuf(x.ID, y.ID)| \geq x.noti\_level$.*

Fact 5 *If $x \xrightarrow{jn} y$ happens, y will send a reply that includes $y.table$ to x immediately. Moreover, each* JN *sent by x includes $x.table$.*

Fact 6 *x sends a message of type* JW *or* JN *to y at most once (x does not send both types of messages to y).*

Fact 7 *By time t_x^e, x has received all of the replies for messages of type* CP, JW, JN, *and* SN *it has sent out.*

Proposition A.1 *Suppose a set of nodes, $W = \{x_1, ..., x_m\}$, $m \geq 1$, join a consistent network $\langle V, \mathcal{N}(V)\rangle$. Consider node x, $x \in W$. Let $u = A(x)$ and let t be the time u sends its positive reply,* JWRly, *to x. Suppose one of the following is true, where $y \in V \cup W$ and $y \neq x$:*

- *$x \xrightarrow{jn} y$ happens;*
- *$y = u$.*

Then if at time t, $\langle y \rightarrow z\rangle_d$, $z \in V \cup W$, and $|csuf(x.ID, z.ID)| \geq x.noti\_level$, then $x \xrightarrow{j} z$ eventually happens.

Proof: Since time t, y can reach z, there must exist a neighbor sequence, $(u_h, u_{h+1}, ..., u_d)$, $h = |csuf(y.ID, z.ID)|$, such that u_h is y, u_d is z, and u_{i+1} is $N_{u_i}(i, z[i])$ for $h \leq i \leq d-1$. Note that the ID of each node in the sequence has suffix $y[h-1]...y[0]$ (which is the same with $z[h-1]...z[0]$).

Next, we prove the following claim: *If $x \xrightarrow{j} u_i$ happens, $h \leq i \leq d-1$, then $x \xrightarrow{j} u_{i+1}$ eventually happens.*

Let $k = x.noti\_level$. To prove the claim, we first need to show that $|csuf(x.ID, u_i.ID)| \geq k$, $h \leq i \leq d-1$. Thus, both $y.ID$ and $z.ID$ have suffix $x[k-1]...x[0]$ (which is equal to $y[k-1]...y[0]$ and $z[k-1]...z[0]$). Let $h_{xy} = |csuf(x.ID, y.ID)|$ and $h_{xz} = |csuf(x.ID, z.ID)|$. Then $h_{xy} \geq k$ (by Facts 3 and 4) and $h_{xz} \geq k$ (by Fact 4). Consequently, $h \geq \min(h_{xy}, h_{xz}) \geq k$. Since all the IDs of u_i, $h \leq i \leq d-1$, have suffix $y[h-1]...y[0]$, they all have suffix $y[k-1]...y[0]$, which is the same with $x[k-1]...x[0]$ according the the above analysis. Hence, $|csuf(x.ID, u_i.ID)| \geq k$, $h \leq i \leq d$.

We then can prove the claim. Suppose $x \xrightarrow{j} u_i$ happens. If $u_{i+1} = u_i$, then the claim holds trivially. If $u_{i+1} \neq u_i$, then let t_i be the time u_i sends its reply to x. If $u_i = u$, then $t_i = t$; If $u_i \neq u$, however, the notification x sends to u_i is *JN*, then $t_i > t$, because x can not send out a *JN* before time t (by Fact 4). Thus, $t_i \geq t$. Since the neighbor sequence between y and z already exists at time t, at t_i, $N_{u_i}(i, z[i])$ is already filled with u_{i+1}. Thus, from u_i's reply, x knows u_{i+1} and will send a *JN* to u_{i+1} if it has not done so (see code in Figures 7 and 10). Hence, the claim holds for any i, $h \leq i \leq d-1$.

Then, given that $x \xrightarrow{j} u_h$ ($u_h = y$) happens, and by induction on i, $h \leq i \leq d-1q$, we conclude that $x \xrightarrow{j} z$ eventually happens. Moreover, x will not change status to *in_system* if it has not received the reply from z. Hence, the time x sends a notification to z is before t_x^e. ■

A.1.1 Single Join

Lemma 5.1 *Suppose node x joins a consistent network $\langle V, \mathcal{N}(V)\rangle$. Then, at time t_x^e, $\langle V \cup \{x\}, \mathcal{N}(V \cup \{x\})\rangle$ is consistent.*

Proof: Suppose $V_{x[k-1]...x[0]} \neq \emptyset$ and $V_{x[k]...x[0]} = \emptyset$, $1 \leq k \leq d-1$. (Such a k must exist since $x.ID$ is unique, thus at least $V_{x[d-1]...x[0]} = \emptyset$ is true.) Since $V_{x[k-1]...x[0]} \neq \emptyset$, $V_{x[i-1]...x[0]} \neq \emptyset$ for all i, $1 \leq i \leq k$.

Initially, x knows g_0, $g_0 \in V$. x then requests to copy level-0 neighbors of g_0. If $k > 0$, then $V_{x[0]} \neq \emptyset$, thus $N_{g_0}(0, x[0]) \neq$ *null*. Let $g_1 = N_{g_0}(0, x[0])$. Next, x requests to copy level-1 neighbors of g_1. Similar to the above argument, if $k > 1$, x will find g_2, $g_2 = N_{g_1}(1, x[1])$. The process is then repeated for level-2, level-3, and so on, and eventually stops at level-k, where $N_{g_k}(k, x[k]) =$ *null* since $V_{x[k]...x[0]} = \emptyset$. At this time, x's c-phase ends. Consider an (i,j)-entry in $x.table$, $0 \leq i \leq k$, $j \in [b]$ and $j \neq x[i]$. If $V_{j \cdot x[k-1]...x[0]} \neq \emptyset$, then given $\langle V, \mathcal{N}(V)\rangle$ is consistent, we know that $N_{g_i}(i,j) \neq$ *null*, where g_i is the node x has requested for its level-i neighbors. By setting $N_x(i,j) = N_{g_i}(i,j)$, it follows that $N_x(i,j) \neq$ *null*. Also, before x turns into an S-node, it sets $N_x(i, x[i]) = x$. Hence, by time t_x^e, table entries in $x.table$ satisfy condition (a) in Definition 3.8.

Next, consider x's w-phase and n-phase. In w-phase, x sends a *JW* to g_k. Since only x is joining, by the time g_k receives the message, g_k knows no other nodes that have suffix $x[k]...x[0]$. g_k then stores x in its table and sends back a positive *JWRly*, which enables x to proceed to n-phase. Let the time g_k sends its reply to x be time t. Given that $\mathcal{N}(V)$ is consistent, at time t, g_k can reach any node v, $v \in V_{x[k-1]...x[0]}$ and $v \neq g_k$. Then by Proposition A.1, $x \xrightarrow{j} v$ eventually happens. Therefore, by time t_x^e, all nodes in $V_{x[k-1]...x[0]}$ have been notified by x and will set

$N_x(k, x[k]) = x$. Now consider any (i, j)-entry in $v.table$, $v \in V$, $i \in [d]$, $j \in [b]$. If $(V \cup \{x\})_{j \cdot v[i-1]...v[0]} \neq \emptyset$, $0 \leq i < k$ (or $i = k$, $j \neq x[i]$), then $V_{v[i-1]...v[0]} \neq \emptyset$, thus $N_v(i, j) = y$, $y \in V_{v[i-1]...v[0]}$. If $(V \cup \{x\})_{j \cdot v[i-1]...v[0]} \neq \emptyset$, $i = k$ and $j = x[i]$, then $N_v(i, j) = x$ by time t^e_x, as proved above. If $(V \cup \{x\})_{j \cdot v[i-1]...v[0]} \neq \emptyset$, $i > k$, then $V_{j \cdot v[i-1]...v[0]} \neq \emptyset$ ($j \cdot v[i-1]...v[0]$ can not be a suffix of $x.ID$), thus, $N_v(i, j) = y$, $y \in V_{v[i-1]...v[0]}$. Altogether, Table entries in $v.table$ also satisfy condition (a) in Definition 3.8 at time t^e. Therefore, $\langle V \cup \{x\}, \mathcal{N}(V \cup \{x\}) \rangle$ is consistent at time t^e. ■

Corollary A.1 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, join a consistent network $\langle V, \mathcal{N}(V) \rangle$. Then for any node x, $x \in W$, if $V_{j \cdot x[i-1]...x[0]} \neq \emptyset$, $i \in [d]$, $j \in [b]$ and $j \neq x[i]$, then there exists a node y, $y \in V_{j \cdot x[i-1]...x[0]}$, such that $N_x(i, j) = y$ at time t^e_x.*

A.1.2 Multiple Joins

In this section, we prove Theorems 1 and 2. To prove Theorem 1, we first prove the lemmas and propositions stated in Section 5.1, as well as some auxiliary lemmas and propositions. To simplify our presentation, we define a suffix $l_i...l_0$ to be empty if $i = 0$.

Lemma 5.2 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, join a consistent network $\langle V, \mathcal{N}(V) \rangle$ sequentially. Then, at time t^e, $\langle V \cup W, \mathcal{N}(V \cup W) \rangle$ is consistent.*

Proof: Proof by induction. Since $\langle V, \mathcal{N}(V) \rangle$ is consistent, by Lemma 5.1, the corollary holds when $j = 1$. Assume the corollary holds for j, $1 \leq j < n$, i.e., by time $t^e_{x_j}$, $\langle V \cup \{x_1, ..., x_j\}, \mathcal{N}(V \cup \{x_1, ..., x_j\}) \rangle$ is consistent. Then x_{j+1} joins at time $t^b_{x_{j+1}}$, where $t^b_{x_{j+1}} > t^e_{x_j}$. Thus, when x_{j+1} joins, $\langle V \cup \{x_1, ..., x_j\}, \mathcal{N}(V \cup \{x_1, ..., x_j\}) \rangle$ is already consistent and there is no other joins during x_{j+1}'s joining period. By Lemma 5.1, by the end of x_{j+1}'s joining period, $\langle V \cup \{x_1, ..., x_{i+1}\}, \mathcal{N}(V \cup \{x_1, ..., x_{i+1}\}) \rangle$ is consistent. Therefore, the lemma holds. ■

Lemma A.1 *Suppose two nodes, x and y, join a consistent network $\langle V, \mathcal{N}(V) \rangle$, $V^{Notify}_x = V_{\omega_1}$ and $V^{Notify}_y = V_{\omega_2}$. Then the joins of x and y are independent iff neither ω_1 nor ω_2 is a suffix of the other.*

Proof: V_{ω_1} is the set of nodes in V with suffix ω_1. Likewise, V_{ω_2} is the set of nodes in V with suffix ω_2. Let $\omega = csuf(\omega_1, \omega_2)$. If neither ω_1 nor ω_2 is a suffix of the other, then $\omega \neq \omega_1$ and $\omega \neq \omega_2$. Thus, $V_{\omega_1} \cap V_{\omega_2} = \emptyset$. By the definition, the joins of x and y are independent. ■

Lemma A.2 *Suppose a set of nodes, $W = \{x_1, ..., x_m\}$, join a consistent network $\langle V, \mathcal{N}(V) \rangle$. For any two nodes x_i and x_j, $x_i \in W$, $x_j \in W$, $x_i \neq x_j$, if $V^{Notify}_{x_i} \cap V^{Notify}_{x_j} = \emptyset$, then $(V \cup W')^{Notify}_{x_i} \cap (V \cup W')^{Notify}_{x_j} = \emptyset$, where $W' \subset W$, $x_i \notin W'$ and $x_j \notin W'$.*

Proof: Proof by contradiction. Assume that $(V \cup W')^{Notify}_{x_i} \cap (V \cup W')^{Notify}_{x_j} \neq \emptyset$.

Suppose $V^{Notify}_{x_i} = V_{\omega_i}$, $(V \cup W')^{Notify}_{x_i} = V_{\omega'_i}$, $V^{Notify}_{x_j} = V_{\omega_j}$, and $(V \cup W')^{Notify}_{x_j} = V_{\omega'_j}$. Then ω_i is a suffix of ω'_i and ω_j is a suffix of ω'_j. (Suppose ω_i is $x_i[k-1]...x_1[0]$, $k \leq d$, then in $V \cup \{x_i\}$, x_i is the only node with suffix $x_i[k]...x[0]$. Also, suppose ω'_i is $x_i[k'-1]...x_1[0]$, $k' \leq d$, then in $V \cup W' \cup \{x_i\}$, x_i is the only node with suffix $x_i[k']...x[0]$. Hence, k' is no less than k.)

Then, by Lemma A.1, either ω'_i or ω'_j is a suffix of the other, since it is assumed that $(V \cup W')^{Notify}_{x_i} \cap (V \cup W')^{Notify}_{x_i} \neq \emptyset$. Without loss of generality, suppose ω'_i is a suffix of ω'_j. It then follows that ω_i is a suffix of ω'_j. Since ω_j is also a suffix of ω'_j, either ω_i or ω_j must be a suffix of the other, which contradicts with Lemma A.1. ■

Corollary A.2 *Suppose a set of nodes, $W = \{x_1, ..., x_m\}$, join a consistent network $\langle V, \mathcal{N}(V) \rangle$. For any two nodes x_i and x_j, $x_i \in W$, $x_j \in W$, $x_i \neq x_j$, if $V^{Notify}_{x_i} \cap V^{Notify}_{x_j} = \emptyset$, then $(V \cup W' \cup \{x_j\})^{Notify}_{x_i} \cap (V \cup W' \cup \{x_i\})^{Notify}_{x_j} = \emptyset$, where $W' \subset W$, $x_i \notin W'$ and $x_j \notin W'$.*

Lemma A.3 *Suppose a set of nodes, $W = \{x_1, ..., x_m\}$, join a consistent network $\langle V, \mathcal{N}(V) \rangle$ independently. For any node x, $x \in W$, if $V_{j \cdot x[i-1]...x[0]} = \emptyset$, $0 \leq i < d-1$, $j \in [b]$ and $j \neq x[i]$, then $(V \cup W')_{j \cdot x[i-1]...x[0]} = \emptyset$, where $W' \subseteq W$.*

Proof: Proof by contradiction. Suppose there exists one node x, $x \in W$, such that $V_{j \cdot x[i-1]...x[0]} = \emptyset$, however, $(V \cup W')_{j \cdot x[i-1]...x[0]} \neq \emptyset$, $j \neq x[i]$. Then there exists a node y, $y \in W'$ such that $y.ID$ has suffix $j \cdot x[i-1]...x[0]$. Thus, $x[i-1]...x[0] = y[i-1]...y[0]$ and $j = y[i]$.

Next, suppose $V^{Notify}_x = x[k_1-1]...x[0]$ and $V^{Notify}_y = y[k_2-1]...y[0]$. (Recall that this indicates $V_{x[k_1-1]...x[0]} \neq \emptyset$ and $V_{y[k_2-1]...y[0]} \neq \emptyset$.) By Lemma A.1, neither $x[k_1-1]...x[0]$ nor $y[k_2-1]...y[0]$ is a suffix of the other.

If $k_2 \geq k_1$, then $x[k_1-1]...x[0] \neq y[k_1-1]...y[0]$ (otherwise, $x[k_1-1]...x[0]$ is a suffix of $y[k_2-1]...y[0]$). On the other hand, $x[i-1]...x[0] = y[i-1]...y[0]$. Thus, $k_1 > i$. Given $V_{x[k_1-1]...x[0]} \neq \emptyset$ and $k_1 > i$, it follows that $V_{x[i-1]...x[0]} \neq \emptyset$. However, $V_{x[i-1]...x[0]} \neq \emptyset$ and $V_{j \cdot x[i-1]...x[0]} = \emptyset$ (i.e. $V_{y[i-1]...y[0]} \neq \emptyset$, $V_{j \cdot y[i-1]...y[0]} \neq \emptyset$ and $j = y[i]$) indicates that $V^{Notify}_y = V_{x[i-1]...x[0]}$. Hence, $k_2 = i$. However, given $k_2 \geq k_1$ and $k_1 > i$, we have $k_2 > i$, which contradicts with $k_2 = i$.

Similarly, if $k_1 \geq k_2$, then $x[k_2-1]...x[0] \neq y[k_2-1]...y[0]$. Again, it must be $k_2 > i$ since $x[i-1]...x[0] = y[i-1]...y[0]$. Given $V_{y[k_2-1]...y[0]} \neq \emptyset$ and $k_2 > i$, it follows that $V_{y[i-1]...y[0]} \neq \emptyset$. Therefore, $V^{Notify}_y = V_{y[i-1]...y[0]}$, since $V_{j \cdot y[i-1]...y[0]} = \emptyset$, $j = y[i]$, but $V_{y[i-1]...y[0]} \neq \emptyset$. Thus, $k_2 = i$, which contradicts with $k_2 > i$. ■

Corollary A.3 *Suppose a set of nodes, $W = \{x_1, ..., x_m\}$, join a consistent network $\langle V, \mathcal{N}(V)\rangle$. Let $G(V_{\omega_1}) = \{x, x \in W, V_x^{Notify} = V_{\omega_1}\}$, $G(V_{\omega_2}) = \{y, y \in W, V_y^{Notify} = V_{\omega_2}\}$. If $V_{\omega_1} \cap V_{\omega_2} = \emptyset$, then for any node x, $x \in G(V_{\omega_1})$, if $V_{j \cdot x[i-1] \ldots x[0]} = \emptyset$, $j \neq x[i]$, then $(V \cup G(V_{\omega_2}))_{j \cdot x[i-1] \ldots x[0]} = \emptyset$.*

Lemma 5.3 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, join a consistent network $\langle V, \mathcal{N}(V)\rangle$ concurrently. If the joins are independent, then at time t^e, $\langle V \cup W, \mathcal{N}(V \cup W)\rangle$ is consistent.*

Proof: Consider any node x, $x \in W$. If $V_{j \cdot x[i-1] \ldots x[0]} \neq \emptyset$ (hence $(V \cup W)_{j \cdot x[i-1] \ldots x[0]} \neq \emptyset$), $i \in [d]$, $j \in [b]$ and $j \neq x[i]$, then by Corollary A.1, at time t^e, $N_x(i, j) = y$, where $y \in V_{j \cdot x[i-1] \ldots x[0]}$. If $V_{j \cdot x[i-1] \ldots x[0]} = \emptyset$, $j \in [b]$ and $j \neq x[i]$, then by Lemma A.3, $(V \cup W)_{j \cdot x[i-1] \ldots x[0]} = \emptyset$, hence $N_x(i, j)$ remains empty. Lastly, $N_x(i, j) = x$, $i \in [b]$ and $j = x[i]$. Therefore, entries in $x.table$ satisfy the consistency condition. [9]

Suppose $V_x^{Notify} = V_{\omega_x}$, $|\omega_x| = k$. Then, similar to the argument in proving Lemma 5.1, it can be shown that by time t^e, x has notified all of the nodes in V_{ω_x}, which in turn have updated the corresponding entries in their table.

The above results are true for any x, $x \in W$. Hence, for any node in v, $v \in V$, if $(V \cup W)_{j \cdot v[i-1] \ldots v[0]} \neq \emptyset$, $i \in [d]$, $j \in [b]$, then $N_v(i, j) = y$, $y \in (V \cup W)_{j \cdot v[i-1] \ldots v[0]}$ by time t^e. Therefore, by time t^e, for any node in the network, the consistency condition holds. We conclude that $\langle V \cup W, \mathcal{N}(V \cup W)\rangle$ is consistent at time t^e. ■

Proposition A.2 *Suppose a set of nodes, $W = \{x_1, ..., x_m\}$, $m \geq 2$, join a consistent network $\langle V, \mathcal{N}(V)\rangle$. For any two nodes x and y, $x \in W$ and $y \in V \cup W$, if $x \overset{j}{\rightarrow} y$ happens, then at time t_x^e, $\langle y \rightarrow x\rangle_d$.*

Proof: Initially, let $i = 0$ and $u_0 = y$. Let the time u_i sends its reply to x be t_i (if $x \overset{jn}{\rightarrow} u_i$, then t_i is the same with the time u_i receives the message from x; if $x \overset{jw}{\rightarrow} u_i$, then t_i is the same with the time u_i receives the message from x if y is an S-node at t_i, or time $t_e^{u_i}$). Also, let $h = |csuf(x.ID, y.ID)|$.

(1) If at time t_i, $N_{u_i}(h_i, x[h_i]) =$ *null*, $h_i = |csuf(x.ID, u_i.ID)|$, then u_i will set $N_{u_i}(h, x[h]) = x$. Hence, $\langle y \rightarrow x\rangle_d$, since a neighbor sequence from y to x, (u_0 (y), u_1, ..., u_i, x), exists.

(2) If at time t_i, $N_{u_i}(h_i, x[h_i]) = v$, $v \neq x$, then from u_i's reply, x finds v in $u_i.table$. Let $u_{i+1} = v$ and $|csuf(x.ID, u_{i+1}.ID)| = h_{i+1}$. Let the time x receives the reply from y be t_i'. If $x \overset{jn}{\rightarrow} u_i$, then x is in n-phase at time t_i' and since $h_{i+1} \geq h_i \geq x.noti\_level$, x needs to send a *JN* to u_{i+1}; If $x \overset{jw}{\rightarrow} u_i$, then x is in w-phase at time t_i' and needs to send u_{i+1} a *JW*. In either case, $x \overset{j}{\rightarrow} u_{i+1}$ happens. Increment i and repeat steps (1) and (2).

We claim that steps (1) and (2) be repeated at most d times, because

- At round i, $h_i > h_{i-1}$.
- At each round i, $h_i \leq d - 1$. The reason is that $x.ID$ is unique in the system, therefore, any other node can share at most $d - 1$ digits (rightmost) with x.

Hence, there exists a node, u_j, $1 \leq i < d - h$, such $N_{u_j}(h_i, x[h_j]) = x$, $h_j = |csuf(x.ID, u_j.ID)|$. Therefore, eventually, there exists a neighbor sequence from y to x, which is (u_0 (y), u_1, ..., u_j, x). Moreover, at time t_x^e, x must have received all replies it expects, which include the reply from u_j. Hence, at time t_x^e, y can already reach x. ■

Proposition A.3 *Suppose a set of nodes, $W = \{x_1, ..., x_m\}$, $m \geq 2$, join a consistent network $\langle V, \mathcal{N}(V)\rangle$. Let x and y be two nodes in W. Suppose there exists a node u, $u \in V \cup W$, such that by time t^e, $x \overset{j}{\rightarrow} u$ has happened, and $y \overset{j}{\rightarrow} u$ or $y \overset{c}{\rightarrow} u$ has happened. If $|csuf(x.ID, y.ID)| = h$ and $x.noti\_level \leq h$, then by time t_{xy}, $t_{xy} = \max(t_x^e, t_y^e)$, there exists a node z, $z \in (V \cup W)_{x[h] \ldots x[0]}$, such that $N_y(h, x[h]) = z$.*

Proof: To prove there exists a node z, $z \in W_{x[h] \ldots x[0]}$, such that $N_y(h, x[h]) = z$ by time t_{xy}, it is sufficient to show that by time t_{xy}, y gets to know at least one node in $W_{x[h] \ldots x[0]}$. (y stores the first node it knows and ignores the others.) Observe that $h > k$ and $x[h] \neq y[h]$.

- Case 1: $|csuf(u.ID, x.ID)| > h$. (For example, $u =$30251, $x =$20251, $y =$10151.)

  In this case, $u \in (V \cup W)_{x[h] \ldots x[0]}$. By the time y sends a notification to u or sends a *CP* to u, y already knows u. Thus, either there already exists a ndoe z, such that $z = N_y(h, x[h])$, or $N_y(h, x[h]) = u$.
- Case 2: $|csuf(u.ID, x.ID)| = h$. (For example, $u =$30151, $x =$20251, $y =$10151.)

  Consider what happens after u receives the message, either a *CP* or a notification, from y. We analyze the case $y \overset{j}{\rightarrow} u$ first. Let $u_0 = u$, and let $i = 0$ initially, then the following process may be repeated:

  - Suppose $|csuf(u_i.ID, y.ID)| = h_i$, $i \geq 0$. If at the time u_i receives the message from y, it sets or has set $N_{u_i}(h_i, y[h_i]) = y$, then let $f = i$ and the process ends; If u_i has already set $N_{u_i}(h_i, y[h_i]) = v$, $v \neq y$, then u_i replies to y with v. Next, y needs to send a notification to v if it has not done so, i.e., $y \overset{j}{\rightarrow} v$ will happen (or has happened). Then, increment i and let $u_i = v$.

[9] By saying that entries in $x.table$ satisfy the consistency condition, or the consistency condition holds at x, we mean that condition (a) in Definition 3.8 is satisfied by each entry in $x.table$.

Similar to the argument in the proof of Proposition A.2, it can be shown that the above process will terminate, i.e., eventually y is stored by a node u_f, $0 \le f \le d - h_0$. Hence, we get a chain of nodes, (u_0, u_1, ..., u_f), where $u_0 = u$ and $u_{i+1} = N_{u_i}(h_i, y[h_i])$ for $0 \le i \le f-1$, $h_i = |csuf(u_i.ID, y.ID)|$.

Similarly, if $y \overset{c}{\to} u$, then there also exists a chain of nodes, (u_0, u_1, ..., u_m), where $u_0 = u$, such that y requests neighbor tables from u_0 to u_j, $0 \le j \le m$, where $u_{i+1} = N_{u_i}(i, y[i])$ for $0 \le i \le j-1$, and then notifies (by sending *JW*) u_j, u_{j+1}, and so on until it is stored by u_f, where $u_{i+1} = N_{u_i}(h_i, y[h_i])$ for $j \le i \le f-1$, $h_i = |csuf(u_i.ID, y.ID)|$. Note that in this case, it is possible that $u_i = u_{i+1}$ for $0 \le i < j$.

In summary, there exists a chain of nodes, (u_0, u_1, ..., u_f), $0 \le m \le d - h_0$, such that $u_0 = u$ and u_{i+1} is a neighbor of u_i, as described above, and either $y \overset{c}{\to} u_i$ or $y \overset{j}{\to} u_i$ happens, $0 \le i \le f$. Note that u_i needs to reply y with $u_i.table$ no matter $y \overset{c}{\to} u_i$ or $y \overset{j}{\to} u_i$ happens. Moreover, $u_i.ID$, $1 \le i \le f$, also has suffix $y[h]...y[0]$, where $y[h-1]...y[0] = x[h-1]...x[0]$ and $y[h] \neq x[h]$, thus, $|csuf(x.ID, u_i.ID)| = h$. Let $u_{f+1} = y$. We call such a chain of nodes, (u_0, u_1, ..., u_f, u_{f+1}), **contact-chain(y, u_0)** (actually, once the chain is established, it is the same with the neighbor sequence from u_0 to y). We then prove the following claim:

Claim A.1 *(Property of contact-chain(y, u_0)) Consider contact-chain(y, u_0), which is ($u_0, u_1, ..., u_f$, u_{f+1}). If after y receives all of the replies from u_0 to u_i and copies nodes from neighbor tables included in the replies,*[10] *$N_y(h, x[h])$ is still empty, then $x \overset{j}{\to} u_{i+1}$ happens eventually,* $0 \le i \le f+1$.

Proof: (of Claim A.1) We prove the above claim by induction on i. In what follows, we say that link (u_i, u_{i+1}) exists at time t, if u_i has stored u_{i+1} in its table by time t.

Base step: If after y receives the reply from u_0 and copies nodes from $u_0.table$, $N_y(h, x[h])$ is still empty, then it indicates that $N_{u_0}(h, x[h]) =$ *null* at the time u_0 sends out the reply to y. Thus, it must be that u_0 does not copy a node z with suffix $x[h]...x[0]$ during its c-phase (otherwise, y would copy z from u_0, since when u_0 replies to y, u_0 must have already finished its c-phase). Let t_1 be the time u_0 sends its reply to y, and t_2 be the time u_0 receives the notification from x.

Then it must be $t_1 < t_2$, as shown in Figure 16. Otherwise, at time t_2, u_0 knows x and at time t_1,

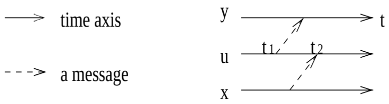


Figure 16. Message sequence chart for Proposition. A.3

$N_{u_0}(h, x[h])$ could not be empty, hence, $N_y(h, x[h])$ could not be empty after y receives u_0's reply. Moreover, link (u_0, u_1) already exists at time t_1, otherwise, u_0 will set y instead of u_1 as the corresponding neighbor. Consequently, link (u_0, u_1) exists at time t_2. Hence, from u_0's reply that includes $u_0.table$, x knows u_1. On receiving the reply from u_0, if x is in n-phase at this point, x will notify u_1 if it has not done so, since $|csuf(x.ID, u_1.ID)| = h$ and $h \ge x.noti\_level$; If x is in w-phase, then the reply from u_0 must be a positive *JWRly* (otherwise, x needs to continue sending *JW* to a node that shares more than h digits with it, which indicates $x.noti\_level > h$), and x then proceeds to n-phase and sends a *JN* to u_1. In either case, $x \overset{j}{\to} u_1$ eventually happens.

Inductive step: Assume the claim holds for all j, $0 \le j \le i$, where $0 \le i \le m-1$. We next prove that if after receiving replies from u_0 to u_{i+1}, $0 \le i \le f-1$, and copying nodes from their neighbor tables, $N_y[h, x[h]$ is still empty, then $x \overset{j}{\to} u_{i+2}$ eventually happens.

By assuming that the claim holds for all j, $0 \le j \le i$, we know that $x \overset{j}{\to} u_{i+1}$ happens. Also, according to the construction of contact-chain(y, u_0), we know that $y \overset{j}{\to} u_{i+1}$ (or $y \overset{c}{\to} u_{i+1}$) happens.

Similar to the base case, let t_1 be the time u_{i+1} sends its reply to y, and t_2 be the time u_{i+1} sends its reply to x. Then it must be that $t_1 < t_2$. Also, we know that link (u_{i+1}, u_{i+2}) already exists at time t_1 as well as at time t_2. Hence, when u_{i+1} replies to x with $u_{i+1}.table$, x knows u_{i+2} and will notify u_{i+2} if it has not done so. Thus, $x \overset{j}{\to} u_{i+2}$ eventually happens. ■

Having proved the claim, it is trivial to show that if after receiving all the replies from u_0 to u_f, $N_y(h, x[h])$ is still empty, then eventually $x \overset{j}{\to} y$ happens (which is before time t_x^e and thus before time t_{xy}). In other words, either that y knows a node in $W_{x[h]...x[0]}$ through u_i, $0 \le i \le f$, or that y receives a notification from x eventually. Therefore, by time t_x^e, there exists a node z, $z \in (V \cup W)_{x[h]...x[0]}$, such that $N_y(h, x[h]) = z$.

- Case 3: $|csuf(u.ID, x.ID)| < h$ (for example, u =30161, x =20251, y =10151).

Let $|csuf(u.ID, x.ID)| = h'$, then $x[h'] = y[h']$, since $x[h-1]...x[0] = y[h-1]...y[0]$ and $h' < h$. In this case, the notification x sends to u must be a *JW*,

[10] See the code in Figures 5 and 8.

since $h' < x.noti\_level$ indicates x can not send a *JN* to u. Moreover, $h' < x.noti\_level$ also indicates that u sends a negative reply to x, otherwise, $x.noti\_level$ would be set to h' (see the code in Figure 7). Then it could be that $N_u(h', x[h']) = y$ or $N_u(h', x[h']) = v$, where $v \neq x$ and $v \neq y$.

If $N_u(h', x[h']) = y$, then from u's reply, x finds y and sends another *JW* to y. Thus, y knows x and will set $N_y(h, x[h]) = x$ if it has not filled that entry.

If $N_u(h', x[h']) = v$, however, $v \neq x$ and $v \neq y$, then let $v_1 = v$ ($|csuf(v_1.ID, x.ID)| > h'$). Let $i = 1$ initially and repeat the following process until $|csuf(v_i.ID, x.ID)| \geq h$:

- Let $h_i = |csuf(v_i.ID, x.ID)|$. If $h_i \geq h$, then according to Case 1 or Case 2, the proposition holds (substitute u with v_i in the proof in Case 1 or Case 2). If $h_i < h$, then, similarly, it can not be $N_{v_i}(h_i, x[h_i]) = x$. Hence, either $N_{v_i}(h_i, x[h_i]) = y$ or that $N_{v_i}(h_i, x[h_i])$ is set to a node other than y. If $N_{v_i}(h_i, x[h_i]) = y$, then $x \xrightarrow{j} y$ eventually happens; otherwise, let $v_{i+1} = N_{v_i}(h_i, x[h_i])$ ($|csuf(v_{i+1}.ID, x.ID)| > |csuf(v_i.ID, x.ID)|$), increment i and repeat this step.

■

As mentioned in Section 5.1, to prove Lemma 5.4, a case in which a set of nodes join a consistent network concurrently and the joins are dependent, we first consider nodes in the same C-set tree and prove that nodes in the same C-set tree and nodes in V can eventually reach each other (stated in Proposition 5.4). Then, we prove Proposition 5.5, which states that nodes in different C-set trees can eventually reach each other. Based on Proposition 5.4 and Proposition 5.5, we present our proof of Lemma 5.4.

Proofs of the following propositions and corollaries are based on induction upon $cset(V, W)$ (the C-set tree realized at time t^e given that nodes in W all have the same noti-set), as defined in Section 3. Propositions 5.1 to 5.4, as well as Propositions A.4 to A.8 make the same assumption as Definition 5.1, namely:

Assumption 5.1 (*for Propositions 5.1 to 5.4 and Propositions A.4 to A.8*)
A set of nodes, $W = \{x_1, ..., x_m\}$, $m \geq 2$, join a consistent network $\langle V, \mathcal{N}(V)\rangle$ concurrently and for any x, $x \in W$, $V_x^{Notify} = V_\omega$, $|\omega| = k$.

For clarity of presentation, we do not repeat the above assumption in the propositions and corollaries that use it. Note that propositions and corollaries after Proposition 5.4 do not make the above assumption any more.

Proposition A.4 *For each node x, $x \in W$, there exists a C-set $C_{l_j...l_1\cdot\omega}$, $1 \leq j \leq d-k$, such that by time t^e, $x \in C_{l_j...l_1\cdot\omega}$, where $l_j...l_1 \cdot \omega$ is a suffix of $x.ID$.*

Proof: Consider an arbitrary node x, $x \in W$. Suppose at the end of c-phase, the nodes x has sent *CP* to are g_0, g_1, ..., and $g_{k'}$, $k \leq k' < d-1$, where at each step, x requests level-i neighbors of g_i, $0 \leq i \leq k'$. Moreover, $g_i = N_{g_{i-1}}(i-1, x[i-1])$ for $1 \leq i \leq k'$. Note that $g_i \in V$ for $0 \leq i \leq k$, while $g_i \in W$ for $k < i \leq k'$. Also, $g_k \in V_\omega$ since g_k shares rightmost k digits (ω) with x. If $k' > k$, by the definition of C-set, $g_{k+1} \in C_{x[k]\cdot\omega}$, $g_{k+2} \in C_{x[k+1]x[k]\cdot\omega}$, ..., and $g_{k'} \in C_{x[k'-1]...x[k]\cdot\omega}$.

Let $g_{k'+1} = N_{g_{k'}}(k', x[k'])$. According to the algorithm, x stops sending *CP* either because that $g_{k'+1}$ is an empty pointer, or that $g_{k'+1}$ is a T-node. First, consider the case that $g_{k'+1}$ is an empty pointer. Hence, from $g_{k'}$'s reply, a *CPRly*, x finds $N_{g_{k'}}(k', x[k']) = null$ and then sends a *JW* to $g_{k'}$. We have shown that $g_{k'} \in C_{x[k'-1]...x[k]\cdot\omega}$. Let $u_1 = g_{k'}$, $k_1 = k'$, and let $i = 1$ initially. Repeat the following process until u_i sets $N_{u_i}(k_i, x[k_i]) = x$.

- If u_i sets $N_{u_i}(k_i, x[k_i]) = x$, then $x \in C_{x[k_i]...x[k]\cdot\omega}$ since $u_i \in C_{x[k_i-1]...x[k]\cdot\omega}$ and $N_{u_i}(k_i, x[k_i]) = x$. The process terminates. If u_i sets $N_{u_i}(k_i, x[k_i]) = v$, $v \neq x$, then $v \in C_{x[k_i]...x[k]\cdot\omega}$. u_i then sends a negative *JWRly* to x and x needs to send another *JW* to v. Let $u_{i+1} = v$ and $k_{i+1} = |csuf(x.ID, u_{i+1}.ID)|$, then $N_{u_{i+1}}(h, x[h]) = u_{i+1}$ for $0 \leq h < k_{i+1}$ (because $u_{i+1}.ID$ has suffix $x_[k_{i+1}-1]...x[0]$), therefore, $u_{i+1} \in C_{x[k_{i+1}-1]...x[k]\cdot\omega}$. Increment i and repeat the process.

As argued in Proposition A.2, the process will terminate since $k' \leq k_i < k_{i+1} \leq d-1$. Eventually, there exists a node u_i, $1 \leq i < d-k'$, such that u_i sets $N_{u_i}(k_i, x[k_i]) = x$ and therefore, $x \in C_{x[k_i]...x[k]\cdot\omega}$

By replacing $u_1 = g_{k'}$ with $u_1 = g_{k'+1}$ in the above arguments, we can get the proof for the case that x stops sending *CP* because $g_{k'+1}$, $g_{k'+1} = N_{g_{k'}}(k', x[k'])$, is a T-node. ■

Proposition 5.1 *If $W_{l_j...l_1\cdot\omega} \neq \emptyset$, $1 \leq j \leq d-k$, $l_j, ..., l_1 \in [b]$, then $C_{l_j...l_1\cdot\omega} \neq \emptyset$.*

Proof: Prove by contradiction. Assume $W_{l_j...l_1\cdot\omega} \neq \emptyset$, however, $C_{l_j...l_1\cdot\omega} = \emptyset$. Consider a node x, $x \in W_{l_j...l_1\cdot\omega}$. By Proposition A.4, by time t^e, there exists a C-set that includes x. Suppose the C-set is $C_{l_i...l_1\cdot\omega}$, then it can not be $i \leq j$. Otherwise, by the definition, x also belongs to set $C_{l_j...l_1\cdot\omega}$, since $l_i...l_1 \cdot \omega$ is a suffix of $l_j...l_1 \cdot \omega$ (both of them are suffixes of $x.ID$). Thus $i > j$. Again, by the definition, $x \in C_{l_i...l_1\cdot\omega}$ implies there exists a node u_{i-1} such that $u_{i-1} \in C_{l_{i-1}...l_1\cdot\omega}$ and $N_{u_{i-1}}(i-1+k, l_i) = x$. Consequently, there must exist a node u_{i-2} such that $u_{i-2} \in C_{l_{i-2}...l_1\cdot\omega}$ and $N_{u_{i-2}}(i-2+k, l_{i-1}) = u_{i-1}$, and so on. Eventually, we get that there must exist a node u_j such that $u_j \in C_{l_j...l_1\cdot\omega}$ and $N_{u_j}(j+k, l_{j+1}) = u_{j+1}$, where $u_{j+1} \in C_{l_{j+1}...l_1\cdot\omega}$. Hence, $C_{l_j...l_1\cdot\omega} \neq \emptyset$, which contradicts with the assumption. ■

Proposition A.5 *Consider any node x, $x \in W$. If $x \in C_{l_{j+1}...l_1\cdot\omega}$ and $x \notin C_{l_j...l_1\cdot\omega}$, $1 \leq j \leq d-k-1$, then*

(a) there exists a node u, $u \in C_{l_j...l_1\cdot\omega}$, such that $N_u(j+k, l_{j+1}) = x$ and u is the node that sends a positive JWRly *to x (i.e. $A(x) = u$);*

(b) $x.noti\_level = j+k$.

If $x \in C_{l_1\cdot\omega}$, then

(a) there exists a node u, $u \in V_\omega$, such that $N_u(k, l_1) = x$ and $A(x) = u$;

(b) $x.noti\_level = k$.

Proof: Following the same argument as in the proof of Proposition A.4, at the end of w-phase of x, we can find a chain of nodes, (g_0, ..., g_k, ..., $g_{k'}(u_1)$, u_2, ..., u_f), where $k \leq k' \leq d-1$, such that x requests level-i neighbors from g_i, $0 \leq i \leq k'$, sends *JW* to u_j, $1 \leq j \leq f$, and finally receives a positive *JWRly* from u_f, $1 \leq f \leq d-k'$. Note that u_f will not reply to x before its status changes to *in_system*. [11]

Then, by the definition of $cset(V, W)$, u_f belongs to a C-set, since each node in the chain, except g_0, is a neighbor of the node proceeds it in the chain. Suppose $|csuf(u_f.ID, x.ID)| = h+k$, $0 \leq h \leq d-1-k$. Hence, $x \in C_{l_{h+k+1}...l_1\cdot\omega}$, where $l_{h+1}...l_1\cdot\omega = x[h+k]...x[0]$, $l_h...l_1\cdot\omega = u_f[h+k-1]...u_f[0]$, and $l_{h+1} \neq u_f[h+k]$. After x receives the positive *JWRly* from u_f, it knows that u_f has set $N_{u_f}(h+k, l_{h+1}) = x$, hence $x \in C_{l_{h+1}...l_1\cdot\omega}$. Thus, x sets $x.noti\_level = h+k$ and proceeds to n-phase to send *JN* to nodes with suffix $x[h+k-1]...x[0]$ (see code in Figure 7).

First, observe that $h \geq j$. Otherwise, since $x \in C_{l_{h+1}...l_1\cdot\omega}$, it is also true that $x \in C_{l_{h'}...l_1\cdot\omega}$, $h' \geq h+1$. It then implies $x \in C_{l_j...l_1\cdot\omega}$, which contradicts with $x \notin C_{l_j...l_1\cdot\omega}$.

Next, we show that $h \leq j$, i.e., $|csuf(u_f.ID, x.ID)| \leq j+k$. We prove this claim by contradiction. Assume $h > j$. Since x only sends *JN* to nodes with suffix $x[h+k-1]...x[0]$ (i.e. suffix $l_h...l_1\cdot\omega$), other nodes can only know x through these nodes. (Note that x would not be a neighbor at any level lower than level-($h+k-1$) in tables of these nodes, since these nodes will fill themselves into the corresponding entries.) Given that $x \in C_{l_{j+1}...l_1\cdot\omega}$ and $x \notin C_{l_j...l_1\cdot\omega}$, there must exist one node y, $y \in C_{l_j...l_1\cdot\omega}$ and $y \neq x$, such that $N_y(j+k, l_{j+1}) = x$ by time t^e (by definition of $cset(V, W)$). Hence, x and y share suffix $l_j...l_1\cdot\omega$. Since x only notifies nodes with suffix $l_h...l_1\cdot\omega$ and $h > j$, it follows that x will not notify y. Then it must be that y knows x through another node, z, where z resides the subtree rooted at $C_{l_h...l_1\cdot\omega}$. There are three possible cases: (i) y copies x from z during c-phase; (ii) y knows x through a reply (a *JWRly* or a *JNRly*) from z or a *JN* from z; (iii) y receives a *SN* informing it about x, which is sent or forwarded by z. Case (i) is impossible, since $N_z(j+k, l_{j+1})$ must be z itself ($z.ID$ also has the suffix $l_{j+1}...l_1\cdot\omega$) and y will set $N_y(j+k, l_{j+1}) = z$. Case (ii) is impossible either, because it indicates that y knows z earlier than it knows x and y would set $N_y(j+k, l_{j+1}) = z$, not x. (See the code in Figures 7, 9 and 10.) Now consider case (iii). If z sends or forwards a *SN* to y, then $|csuf(x.ID, y.ID)| > |csuf(x.ID, z.ID)|$, since both $x.ID$ and $y.ID$ have the same desired suffix of an entry in $z.table$. However, we know that $|csuf(x.ID, y.ID)| < |csuf(x.ID, z.ID)|$, because $|csuf(x.ID, y.ID)| = j+k$, $|csuf(x.ID, z.ID)| = h+k$ and $h > j$. Therefore, case (iii) is also impossible. Thus, we conclude that $h \leq j$.

Combining the two results, $h \geq j$ and $h \leq j$, we get $h = j$. Hence, $u_f \in C_{l_j...l_1\cdot\omega}$ and $x.noti\_level = j+k$. ■

Proposition 5.2 *Let u be a node in V_ω. If $W_{l_1\cdot\omega} \neq \emptyset$, $l_1 \in [b]$, then there exists a node x, $x \in W_{l_1\cdot\omega}$, such that $N_u(k, l_1) = x$ by time t^e.*

Proof: By Proposition 5.1, if $W_{l\cdot\omega} \neq \emptyset$, $C_{l\cdot\omega} \neq \emptyset$. Let x be a node in $C_{l\cdot\omega}$ (thus x is also in $W_{l\cdot\omega}$). By Proposition A.5, there exists a node y, $y = A(x)$ and $y \in V_\omega$. Thus, if $u = y$, $N_u(k, l) = x$. Next, consider the case $u \neq y$. Proposition A.5, $x.noti\_level = k$. Hence, x would notify nodes with suffix $x[k-1]...x[0]$, where $x[k-1]...x[0] = \omega$. Since $\mathcal{N}(V)$ is consistent before x joins, there exists a neighbor sequence between y and any other node in V_ω. By Proposition A.1, for any v, $v \in V_\omega$, $v \neq y$, $x \xrightarrow{j} v$ eventually happens. Therefore, $x \xrightarrow{j} u$ eventually happens. When u receives the notification from x, if $N_u(k, l) = null$, then u sets $N_u(k, l) = x$ if it has not filled that entry. ■

Corollary A.4 *Let u be a node in V_ω. If $W_{l\cdot\omega} \neq \emptyset$, $l \in [b]$, then there exists a node x, $x \in W_{l\cdot\omega}$, such that by time t^e, $x \xrightarrow{j} u$ has happened.*

Proposition A.6 *For any C-set, $C_{l_j...l_1\cdot\omega}$, in $cset(V, W)$, $1 \leq j \leq d-k$, l_1,...,$l_j \in [b]$, the following assertions hold:*

(a) If $|C_{l_j...l_1\cdot\omega}| > 1$, then for any two nodes, x and y, $x \in C_{l_j...l_1\cdot\omega}$, $y \in C_{l_j...l_1\cdot\omega}$, $x \neq y$, by time t_{xy}, $t_{xy} = \max(t^e_x, t^e_y)$, at least one of $x \xrightarrow{j} y$ and $y \xrightarrow{j} x$ has happened. Moreover, at time t_{xy}, $\langle x \rightarrow y\rangle_d$ and $\langle y \rightarrow x\rangle_d$.

(b) For each x, $x \in C_{l_j l_{j-1}...l_1\cdot\omega}$, if $W_{l\cdot l_{j-1}...l_1\cdot\omega} \neq \emptyset$, $l \in [b]$ and $l \neq l_j$, then there exists a node y, $y \in$

[11] The chain of nodes could also be (g_0, ..., g_k, ..., $g_{k'}$, $g_{k'+1}(u_1)$, u_2, ..., u_m), $1 \leq m \leq d-k'-1$, where $g_{k'+1}$ is still a T-node when x copies and constructs its table at level-k'. The proof in this case is quite similar, so we omit it here.

$W_{l \cdot l_{j-1} \ldots l_1 \cdot \omega}$, such that $N_x(k+j-1, l) = y$ by time t^e.

Proof: Part(a) of claims that if $x \in C_{l_j \ldots l_1 \cdot \omega}$, then it can reach another node in the same C-set by the time both of them are S-nodes. Part(b) claims that if $x \in C_{l_j \ldots l_1 \cdot \omega}$, then for any sibling C-set of $C_{l \cdot l_{j-1} \ldots l_1 \cdot \omega}$, x eventually stores a node with the suffix of that sibling C-set. We prove the proposition by induction on j.

Base step: $j = 1$. Consider nodes x and y, $x \in C_{l_1 \cdot \omega}$ and $y \in C_{l \cdot \omega}$, $l_1 \in [b]$, $l \in [b]$ (l may or may not be the same with l_1), and $x \neq y$. By the definition of $cset(V, W)$, by time t^e, there exists a node u_x, $u_x \in V_\omega$, such that $N_{u_x}(k, l_1) = x$. Likewise there exists a node u_y, $u_y \in V_\omega$, such that $N_{u_y}(k, l) = y$. By Proposition A.5, $x.noti\_level = y.noti\_level = k$. Also, by Corollary A.4, $x \xrightarrow{j} u_y$ happens. Likewise, $y \xrightarrow{j} u_x$ happens.

(i) Suppose $l = l_1$, i.e., x and y belong to the same C-set. Then at the time u_y receives the notification from x, $N_{u_y}(k, l_1)$ is set to y already (otherwise, u_y would set $N_{u_y}(k, l_1) = x$, since $x.ID$ also has the suffix $l_1 \cdot \omega$). Then, from u_y's reply, x knows y and will notify y, since $|csuf(x.ID, y.ID)| \geq |l_1 \cdot \omega| > k$. Hence, $x \xrightarrow{j} y$ happens. Then, by Proposition A.2, by time t^e_x, $\langle y \rightarrow x \rangle_d$. Similarly, $y \xrightarrow{j} x$ eventually happens and by time t^e_y, $\langle x \rightarrow y \rangle_d$. Thus, part(a) holds when $j = 1$.

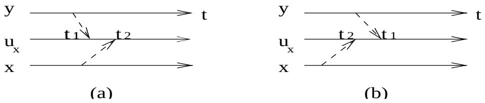


Figure 17. Message sequence chart for base case

(ii) Suppose $l \neq l_1$. Let t_1 be the time u_x receives the notification from y, and t_2 be the time u_x receives the notification from x. If $t_1 < t_2$, as shown in Figure 17(a), then at time t_1, either u_x sets $N_{u_x}(k, l) = y$ or u_x has set $N_{u_x}(k, l) = v$, $v \in W_{l \cdot \omega}$ and $v \neq y$. In either case, at time t_2, $N_{u_x}(k, l) \neq$ *null* and from u_x's reply, x knows node $N_{u_x}(k, l)$ and sets $N_x(k, l) = N_{u_x}(k, l)$. If $t_1 > t_2$, as shown in Figure 17(b), then y knows x from u_x's reply and will notify x if it has not done so. x then sets $N_x(k, l) = y$ if it has not filled that entry. Hence, part (b) also holds when $j = 1$.

Inductive step: In this step, we prove that if the proposition holds at j, then it also holds at $j+1$, $1 \leq j \leq d-k-1$.

Consider node x, $x \in C_{l_{j+1} \ldots l_1 \cdot \omega}$. Then x may or may not belong to $C_{l_j \ldots l_1 \cdot \omega}$. We call the case $x \notin C_{l_j \ldots l_1 \cdot \omega}$ **Case 1**, and call the case $x \in C_{l_j \ldots l_1 \cdot \omega}$ **Case 2**. We consider the two cases separately.

- **Case 1:** $x \in C_{l_{j+1} \ldots l_1 \cdot \omega}$ and $x \notin C_{l_j \ldots l_1 \cdot \omega}$.
  - **1.a** In this case, we need to prove part(a) of the proposition holds. If $|C_{l_{j+1} \ldots l_1 \cdot \omega}| > 1$, then consider any node y, $y \in C_{l_{j+1} \ldots l_1 \cdot \omega}$ and $y \neq x$:
    - ∗ **1.a.1** $y \notin C_{l_j \ldots l_1 \cdot \omega}$;
    - ∗ **1.a.2** $y \in C_{l_j \ldots l_1 \cdot \omega}$.
  - **1.b** In this case, we need to prove part(b) of the proposition holds. Consider the following two cases:
    - ∗ **1.b.1** $\exists y, y \in C_{l \cdot l_j \ldots l_1 \cdot \omega} \wedge y \notin C_{l_j \ldots l_1 \cdot \omega}$;
    - ∗ **1.b.2** $\forall y, y \in C_{l \cdot l_j \ldots l_1 \cdot \omega} \Rightarrow y \in C_{l_j \ldots l_1 \cdot \omega}$.
- **Case 2:** $x \in C_{l_{j+1} \ldots l_1 \cdot \omega}$ and $x \in C_{l_j \ldots l_1 \cdot \omega}$.
  - **2.a** To prove part(a) of the proposition holds, consider any node y, $y \in C_{l_{j+1} \ldots l_1 \cdot \omega}$ and $y \neq x$:
    - ∗ **2.a.1** $y \notin C_{l_j \ldots l_1 \cdot \omega}$;
    - ∗ **2.a.2** $y \in C_{l_j \ldots l_1 \cdot \omega}$.
  - **2.b** To prove part(b) of the proposition holds, consider the following cases:
    - ∗ **2.b.1** $\exists y, y \in C_{l \cdot l_j \ldots l_1 \cdot \omega} \wedge y \in C_{l_j \ldots l_1 \cdot \omega}$;
    - ∗ **2.b.2** $\forall y, y \in C_{l \cdot l_j \ldots l_1 \cdot \omega} \Rightarrow y \notin C_{l_j \ldots l_1 \cdot \omega}$.

First, we prove the following claim:

Claim A.2 *Suppose Proposition A.6 holds at j, $1 \leq j \leq d-k-1$. If $x \in C_{l_{j+1} \ldots l_1 \cdot \omega}$, $y \in C_{l \cdot l_j \ldots l_1 \cdot \omega}$, $l \in [b]$, however, $x \notin C_{l_j \ldots l_1 \cdot \omega}$ and $y \notin C_{l_j \ldots l_1 \cdot \omega}$. Then at least one of $x \xrightarrow{j} y$ and $y \xrightarrow{j} x$ eventually happens.*

Proof: By Proposition A.5, there exists a node u_x, $u_x \in C_{l_j \ldots l_1 \cdot \omega}$, such that $u_x = A(x)$. Likewise, there exists a node u_y, $u_y \in C_{l_j \ldots l_1 \cdot \omega}$, such that $u_y = A(y)$. Figure 18(a) and (b) illustrate the relationship of the four nodes, where in Figure 18(a), $l = l_{j+1}$, and in Figure 18(b), $l \neq l_{j+1}$.

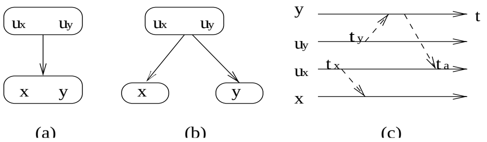


Figure 18. C-sets and message sequences, case 1.a.1 and case 1.b.1

Let the time u_x sends the positive *JWRly* to x be t_x, and the time u_y sends the positive *JWRly* to y be t_y. Without loss of generality, suppose $t_x < t_y$, as shown in Figure 18(c). Then at time t_y, both u_x and u_y are already S-nodes (by Fact 2). Since it is assumed that the proposition holds at j, by part(a) of the proposition, by time t_y, u_x and u_y already can reach each other. Hence, by the time y receives the reply from u_y, u_x and u_y can reach each other. By Proposition A.1, $y \xrightarrow{j} u_x$ eventually happens. Suppose u_x receives the notification from y at time t_a, clearly, $t_a > t_y$, hence, $t_a > t_x$. Then, from u_x's reply, y knows x and will notify x if it has not done so. Thus, $y \xrightarrow{j} x$

eventually happens. Likewise, if $t_y < t_x$, then $x \xrightarrow{j} y$ eventually happens. Hence, at least one of $y \xrightarrow{j} x$ and $x \xrightarrow{j} y$ eventually happens. ■

Based on the claim, we next prove that in case 1.a.1, part(a) of the proposition holds at $j+1$, and in case 1.b.1, part(b) holds at $j+1$. We use the same notation of u_x, u_y, t_x and t_y as above in proving the proposition in these two cases.

- Case 1.a.1. Without loss of generality, suppose $t_x < t_y$. By the proof of Claim A.2, $y \xrightarrow{j} x$ eventually happens. By Proposition A.2, $\langle x \to y\rangle_d$ by time t_y^e. Next, we need to show $\langle y \to x\rangle_d$ by time t_{xy}, which we will prove by first showing that $x \xrightarrow{j} u_y$ happens during x's n-phase.

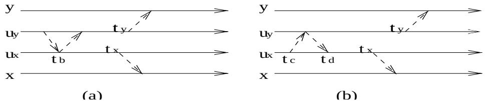


Figure 19. Message sequence chart for case 1.a.1

By assuming that the proposition holds at j, we know that at least one of $u_x \xrightarrow{j} u_y$ and $u_y \xrightarrow{j} u_x$ happens.

(i) First, suppose $u_y \xrightarrow{j} u_x$ happens. Let t_b be the time u_x sends its reply to u_y. Then, it must be that $t_b < t_y$ (by Facts 1 and 2). Also, it must be $t_b > t_x$, as shown by Figure 19(a). Otherwise, x would be included in u_x's reply to u_y and u_y would set $N_{u_y}(j+k, l_{j+1}) = x$, which contradicts with $N_{u_y}(j+k, l_{j+1}) = y$. Then, consider the *contact-chain(u_x, u_y)* (as defined in the proof of Proposition 5.2, Case 2), (u_0, u_1, ..., u_f, u_{f+1}), where $u_0 = u_x$, $u_{f+1} = u_y$, and u_{i+1} is a neighbor of u_i for all i. (Actually, this chain of nodes is also the neighbor sequence from u_x to u_y.) Then after receiving replies from all u_i, $0 \le i \le f$, $N_{u_i}(j+k, l_{j+1})$ must still be empty (no node except u_y knows y before time t_y, hence, u_y can not copy y from any other node). By Claim A.1, eventually, $x \xrightarrow{j} u_y$ happens.

(ii) Second, suppose $u_x \xrightarrow{j} u_y$ happens. Let t_c be the time u_y receives the notification from u_x, and t_d be the time u_x receives the reply from u_y. Clearly, $t_c < t_d$. Moreover, by Facts 1 and 2, $t_d < t_x$. Figure 19(b) depicts the order of the events. If at time t_x, u_y is already an S-node, then by Proposition A.1, eventually $x \xrightarrow{j} u_y$ happens. If at t_x, u_y is not an S-node yet, then consider time t_c again. Since u_x has already known u_y at t_c, either u_x stores (or has stored) u_y in its table at t_c, or that u_x has set another node, v, $v \ne u_y$, at the corresponding table entry. In the first case, x will know u_y from u_x's reply and notify u_y. In the second case, u_y knows v from u_x's notification that includes $u_x.table$. Since u_y is still a T-node on receiving u_x's notification, it needs to notify v if it has not done so. Also, x knows v from u_x's reply and needs to notify v too. Now consider the contact-chain(u_y, v). Similarly to the argument in case (i), $N_{u_y}(j+k, l_{j+1})$ must still be empty after it receives replies from all other nodes in the chain. Hence, by Claim A.1, eventually $x \xrightarrow{j} u_y$ happens. Therefore, if $u_x \xrightarrow{j} u_y$ happens, $x \xrightarrow{j} u_y$ will happen.

Hence, no matter whether $u_y \xrightarrow{j} u_x$ or $u_x \xrightarrow{j} u_y$ happens, $x \xrightarrow{j} u_y$ will happen. Then, by the time u_y receives the notification from x, it must already set $N_{u_y}(j+k, l_{j+1}) = y$ (otherwise, it would fill x into that entry). Hence, x knows y from u_y's reply and will notify y if it has not done so. By Proposition A.2, it then follows that $\langle y \to x\rangle_d$ by time t_x^e.

If $t_x > t_y$, by reversing the role of x and y in the above arguments, we can get the same conclusion. Therefore, part(a) holds at $j+1$ in case 1.a.1.

- Cases 1.b.1 Consider a node y, $y \in C_{l \cdot l_j \ldots l_1 \cdot \omega}$ and $y \notin C_{l_j \ldots l_1 \cdot \omega}$. By Claim A.2, at least one of $y \xrightarrow{j} x$ and $x \xrightarrow{j} y$ eventually happens. Hence, by the time x receives the notification from y (or the time x sends a notification from y), x knows y. Then, by time t^e, $N_x(j+k, l)$ can not be empty. Therefore, part(b) holds at $j+1$ in case 1.b.1.

Next, we consider cases 1.a.2 and 1.b.2, where $x \in C_{l_{j+1} \ldots l_1 \cdot \omega}$ but $x \notin C_{l_j \ldots l_1 \cdot \omega}$, however, $y \in C_{l \cdot l_j \ldots l_1 \cdot \omega}$ and $y \in C_{l_j \ldots l_1 \cdot \omega}$, $l \in [b]$. (In case 1.b.2, we consider an arbitrary node y, $y \in C_{l \cdot l_j \ldots l_1 \cdot \omega}$.) Let $u_x = A(x)$, thus $u_x \in C_{l_j \ldots l_1 \cdot \omega}$ (by Corollary A.5).

Since u_x and y both belong to $C_{l_j \ldots l_1 \cdot \omega}$, at least one of $u_x \xrightarrow{j} y$ and $y \xrightarrow{j} u_x$ eventually happens (by assuming that the proposition holds at j). Let t_1 be the time u_x sends the positive *JWRly* to x, and t_2 be the time u_x receives the notification from y if $y \xrightarrow{j} u_x$ happens, otherwise, let t_2 be the time u_x sends a notification to y.

- Case 1.a.2. In this case, $l = l_{j+1}$, as shown in Figure 20(a). Then, $u_x \xrightarrow{j} y$ can not happen, because if it happens, the time u_x sends a notification to y is earlier than t_1 and u_x would set $N_{u_x}(j+k, l_{j+1})$ to be y instead of x. Hence, $y \xrightarrow{j} u_x$ happens, which indicates $t_1 < t_2$, otherwise, at time t_2, u_x would set $N_{u_x}(j+k, l_{j+1}) = y$. Figure 20(b) depicts the order of events. Hence, when u_x replies to y with $u_x.table$, x is included in $u_x.table$. Then, y knows x from u_x's reply and $y \xrightarrow{j} x$ will happen. By Proposition A.2, $\langle x \to y\rangle_d$ holds by time t_y^e (and hence by time t_{xy}, where $t_{xy} = \max(t_x^e, t_y^e)$). Next, we need to show that by time t_{xy}, $\langle y \rightarrow x \rangle_d$.

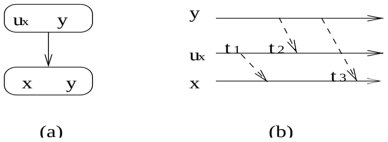


Figure 20. C-sets and message sequences, case 1.a.2

Since $y \overset{j}{\rightarrow} x$, y knows x.

- If y stores x in its table, then it follows trivially $\langle y \rightarrow x \rangle_d$ by time t_y^e.
- If y has already set z as the corresponding neighbor (i.e., $N_y(h_{xy}, x[h_{xy}]) = z$, $h_{xy} = |csuf(x.ID, y.ID)|$), then from y's notification that includes $y.table$, x knows z. Let the time x receives the notification from y be t_3. Note that z share more digits with x than y does.

  (i) If at time t_3, x is still a T-node, then x will notify z if it has not done so. By Proposition A.2, by time t_x^e, z can reach x. Then, by time t_x^e, there exists a neighbor sequence from z to x, say, (u_h, u_{h+1}, u_{h+2}, ..., x), where $h = |csuf(z.ID, x.ID)|$, $u_h = z$, and $u_{h'+1} = N_{u_{h'}}(h', x[h'])$ for $h \leq h' \leq d$. Therefore, by time t_x^e, there exists a neighbor sequence from y to x, which is (y, u_h, u_{h+1}, ..., x).

  (ii) If at time t_3, x is already an S-node, then by the algorithm, after receiving x's reply, y will send a *SN*(y, x) to z to inform z (see the code in Figures 9 and 10). If z already sets z_1 as the corresponding neighbor, $z_1 \neq x$, it would then forward *SN*(y, x) to z_1. The message is forwarded until eventually some node stores or has already stored x and sends a *SNRly* to y. Note that each receiver of the message shares at least one more digits with x than the sender (or forwarder) of that message does. Hence, by time t_y^e (by Fact 7, y has received the *SNRly* by t_y^e), a neighbor sequence (z, z_1, ..., x) exists, thus a neighbor sequence exists from y to x, which is (y, z, z_1, ..., x).

  Therefore, in either case, $\langle y \rightarrow x \rangle_d$ by time t_{xy}. Hence, part(a) of the proposition holds in case 1.a.2.

- Case 1.b.2. In this case, $l \neq l_{j+1}$, as shown in Figure 21(a). Consider any node y, $y \in C_{l \cdot l_j \ldots l_1 \cdot \omega}$. We use the same notation of u_x, t_1 and t_2 as in the proof of case 1.a.2.

  If $t_1 < t_2$, then t_2 is the time that u_x receives the notification from y, as shown in Figure 21(b). (By Facts 1 and 2, if t_2 is the time that u_x sends a notification to y, then $t_2 < t_1$.)

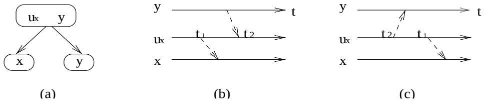


Figure 21. C-sets and message sequences, case 1.b.2

  Then, y knows x from u_x's reply and will notify x. Hence, by time t_{xy}, both x and y has known each other. Hence, $N_x(j+k, l)$ can not be empty by time t_{xy}, and neither can $N_y(j+k, l_{j+1})$ be empty.

  If $t_1 > t_2$ and $y \overset{j}{\rightarrow} u_x$ happens, then Proposition A.3 can be utilized Recall that $x \in C_{l_{j+1} l_j \ldots l_1 \cdot \omega}$, $y \in C_{l \cdot l_j \ldots l_1 \cdot \omega}$, $l \neq l_{j+1}$. Hence, $|csuf(x.ID, y.ID)| = j + k$. Moreover, by Corollary A.5, $x.noti\_level = j + k$ and $y.noti\_level \leq j + k - 1$. Also, we know $u_x \in C_{l_j \ldots l_1 \cdot \omega}$ and $u_x[j+k] \neq l_{j+1}$ (otherwise, $N_{u_x}(k+j, l_{j+1})$ is u_x itself, not x), thus $|csuf(x.ID, u_x.ID)| = j + k$. Given the above facts and that $x \overset{j}{\rightarrow} u_x$ happens (x sends a *JW* to u_x), by Proposition A.3, there exists a node x', $x' \in W_{l_{j+1} \ldots l_1 \cdot \omega}$, such that $N_y(j+k, l_{j+1}) = x'$ by time t_{xy}. Likewise, there exists a node y', $y' \in W_{l \cdot l_j \ldots l_1 \cdot \omega}$, such that $N_x(j+k, l) = y'$ by time t_{xy}.

  If $t_1 > t_2$ and $u_x \overset{j}{\rightarrow} y$ happens, as shown in Figure 21(c), then, at time t_2, u_x is still a T-node (by Fact 1).

  - If at time t_2, u_x sets $N_{u_x}(j+k, l) = y$, then at time t_1, y is already in $u_x.table$. Hence, from u_x's reply, x copies y and will also notify y (see the code in Figure 7). Hence, by time t_{xy}, neither $N_x(j+k, l)$ nor $N_y(j+k, l_{j+1})$ is empty.
  - If at time t_2, $N_{u_x}(j+k, l) = z$, $z \neq y$, and y is already an S-node at time t_1, then from u_x's reply, x copies z. Thus, by time t_{xy}, $N_x(j+k, l) = z$. Next, we prove that there exists a node v, $v \in W_{l_{j+1} \ldots l_1 \cdot \omega}$, such that $N_y(j+k, l_{j+1}) = v$ by time t_{xy}. Observe that at time t_1, both u_x and y are S-nodes, and by assuming the proposition holds at j, we know that by time t_1, u_x and y can already reach each other. Then, by Proposition A.1, $x \overset{j}{\rightarrow} y$ eventually happens, which guarantees that by t_{xy}, $N_y(j+k, l_{j+1})$ is not empty.
  - If at time t_2, $N_{u_x}(j+k, l) = z$, $z \neq y$, however, y is still a T-node at time t_1, then similar to the above case, x copies z from u_x's reply. Thus, by time t_{xy}, $N_x(j+k, l) = z$. Next, consider $N_y(j+k, l_{j+1})$. From u_x's notification, which includes $u_x.table$ that already includes z, y knows z. Since y is still a T-node and $y.noti\_level \leq |csuf(y.ID, z.ID)|$ ($|csuf(y.ID, z.ID)| \geq j + k$ and $y.noti\_level \leq j - 1 + k$), y will notify z if it has not done so. Similarly, x will

notify z ($|csuf(x.ID, z.ID)| = j + k$ and $x.noti\_level = j + k$). Hence, both $y \xrightarrow{j} z$ and $x \xrightarrow{j} z$ will happen. By Proposition A.3, there exists a node x', $x' \in W_{l_{j+1} \dots l_1 \cdot \omega}$, such that $N_y(j + k, l_{j+1}) = x'$ by time t_{xy}.

Therefore, part(b) holds in case 1.b.2.

Finally, we consider cases 2.a.1, 2.a.2, 2.b.1 and 2.b.2.

- Case 2.a.1. This case is symmetric to case 1.a.2, hence by reversing roles of x and y in the proof of case 1.a.2, we can prove that part(a) holds in case 2.a.1.
- Case 2.a.2. In this case, both x and y also belong to $C_{l_j \dots l_1 \cdot \omega}$. By assuming the proposition holds at j, we know that at least one of x and y will send a notification to the other and by time t_{xy}, they can reach each other. Hence part(a) holds trivially in case 2.a.2.
- Case 2.b.1. Consider a node y, $y \in C_{l \cdot l_j \dots l_1 \cdot \omega}$ and $y \in C_{l_j \dots l_1 \cdot \omega}$. Then x and y can reach each other by t_{xy}, $t_{xy} = \max(t_x^e, t_y^e)$, by assuming that the proposition holds at j. Consequently, neither $N_x(j + k, l)$ nor $N_y(j + k, l_{j+1})$ could be empty at time t_{xy}. Therefore, part(b) holds in case 2.b.1.
- Case 2.b.2. Consider an arbitrary y, $y \in C_{l \cdot l_j \dots l_1 \cdot \omega}$. Then, by reversing roles of x and y in the proof of case 1.b.2, we can prove that part(b) holds in case 2.b.2. (In the proof of case 1.b.2, we have proved that by time t_{xy}, neither $N_x(l + k, l)$ nor $N_y(l + k, l_{j+1})$ is empty.)

■

Corollary A.5 *If $x \in C_{l_j \dots l_1 \cdot \omega}$ and $C_{l \cdot l_{j-1} \dots l_1 \cdot \omega} \neq \emptyset$, $1 \leq j \leq d - k$, $l \in [b]$ and $l \neq l_j$, then at least one of the following assertions is true:*

1. *There exists a node y, $y \in C_{l \cdot l_{j-1} \dots l_1 \cdot \omega}$, such that $y \xrightarrow{j} x$ has happened by t^e.*
2. *Let $y = N_x(j - 1 + k, l)$. Then x stores y in $x.table$ before time t_x^e.*

Proof: Proof of the corollary is implied by the proof of Proposition A.6. To prove that the second assertion is true, it is sufficient to show that x knows a node in $W_{l \cdot l_{j-1} \dots l_1 \cdot \omega}$ before time t_x^e.

First, consider the base case, $x \in C_{l_1 \cdot \omega}$. Let y be a node in $C_{l \cdot \omega}$, $l \in [b]$ and $l \neq l_1$. In the proof of Proposition A.6, we have shown that in this case both $y \xrightarrow{j} x$ and $x \xrightarrow{j} y$ will happen. Hence, the corollary holds in the base case.

Next, consider node x, $x \in C_{l_j \dots l_1 \cdot \omega}$, $2 \leq j \leq d - k$.

- Suppose $x \notin C_{l_{j-1} \dots l_1 \cdot \omega}$. Let u_x be the attaching-node of x, $u_x \in C_{l_{j-1} \dots l_1 \cdot \omega}$. Then, consider $C_{l \cdot l_{j-1} \dots l_1 \cdot \omega}$, $l \in [b]$ and $l \neq l_1$.

  If there exists a node y, such that $y \in C_{l \cdot l_{j-1} \dots l_1 \cdot \omega}$ and $y \notin C_{l_{j-1} \dots l_1 \cdot \omega}$, then let u_y be the attaching-node of y. Moreover, let t_x be the time u_x sends its positive reply (*JWRly* to x, and t_y be the time u_y send its positive reply (*JWRly* to y. Then, according to the proof of Proposition A.6 (in the part of considering cases 1.a.1 and 1.b.1 together), if $t_x < t_y$, then $y \xrightarrow{j} x$ will happen; if $t_x > t_y$, then $x \xrightarrow{j} y$ will happen. If $y \xrightarrow{j} x$ happens, the corollary holds; If $x \xrightarrow{j} y$ happens, then x knows y, $y \in W_{l \cdot l_{j-1} \dots l_1 \cdot \omega}$, before time t_x^e. Thus, in either case, the corollary holds.

  If for each node in $C_{l \cdot l_{j-1} \dots l_1 \cdot \omega}$, it also belongs to $C_{l_{j-1} \dots l_1 \cdot \omega}$, then pick any node y, $y \in C_{l \cdot l_{j-1} \dots l_1 \cdot \omega}$ (the corresponding case in the proof of Proposition A.6 is case 1.b.2). By part(a) of Proposition A.6, either $u_x \xrightarrow{j} y$ or $y \xrightarrow{j} u_x$ happens.

  - If $y \xrightarrow{j} u_x$ happens, let the time u_x receives the notification from y be t_1. If $t_x < t_1$, then y knows x from u_x's reply and $y \xrightarrow{j} x$ will happen. If $t_x > t_1$, then since u_x knows y at time t_1, $N_{u_x}(j - 1 + k, l) \neq$ *null* at t_1, hence $N_{u_x}(j - 1 + k, l) \neq$ *null* at t_x. Then, from u_x's reply, x knows a node z, $z = N_{u_x}(j - 1 + k, l)$ and $z \in W_{l \cdot l_{j-1} \dots l_1 \cdot \omega}$. Thus, $N_x(j - 1 + k, l) \neq$ *null* by time t_x^e.
  - If $u_x \xrightarrow{j} y$ happens, then when u_x replies to x, $N_{u_x}(j - 1 + k, l) \neq$ *null* and hence, x is able to store a node in its ($j - 1 + k, l$)-entry by time t_x^e.

- Suppose $x \in C_{l_{j-1} \dots l_1 \cdot \omega}$. Consider any non-empty C-set, $C_{l \cdot l_{j-1} \dots l_1 \cdot \omega}$, $l \in [b]$ and $l \neq l_1$.

  If there exists a node y, $y \in C_{l \cdot l_{j-1} \dots l_1 \cdot \omega}$ and $y \in C_{l_{j-1} \dots l_1 \cdot \omega}$, then by part(a) of Proposition A.6, either $y \xrightarrow{j} x$ or $x \xrightarrow{j} y$ happens. Hence, the corollary holds.

  If every node in $C_{l \cdot l_{j-1} \dots l_1 \cdot \omega}$ does not belong to $C_{l_{j-1} \dots l_1 \cdot \omega}$, then pick a node y, $y \in C_{l \cdot l_{j-1} \dots l_1 \cdot \omega}$. (The corresponding case in the proof of Proposition A.6 is case 2.b.1.) Let u_y be the attaching-node of y. By part(b) of Proposition A.6, either $u_y \xrightarrow{j} x$ or $x \xrightarrow{j} u_y$ happens. Let t_y be the time u_y replies to y, and t_x be the time u_y sends its notification to x if $u_y \xrightarrow{j} x$ happens, otherwise, let t_x be the time u_y receives x's notification.

  - If $u_y \xrightarrow{j} x$ happens, then $t_x < t_y$. (i) If at time t_y, x is already an S-node, then by part(a) of Proposition A.6, u_y can reach x at time t_y. By Proposition A.1, $y \xrightarrow{j} x$ will happen. (ii) If at time t_y, x is still a T-node and $N_{u_y}(j - 1 + k, l_j) = x$, then y knows x from u_y's reply and $y \xrightarrow{j} x$ will happen. (iii) If at time t_y, x is still a T-node and u_y $N_{u_y}(j - 1 + k, l_j) = x'$, $x' \neq x$, then from u_y's notification, which includes $u_y.table$, x knows x' and will notify x' if it has not done so, i.e., $x \xrightarrow{j} x'$ eventually happens. Likewise, from u_y's reply, y knows x' and will notify x' too, i.e., $y \xrightarrow{j} x'$ eventually happens. Consider the *contact-chain(x', x)*, as defined in the proof of Proposition A.3, then by Claim A.1, either that by the time x receives replies from all other nodes in the chain, x has stored a node in $(j-1+k, l)$-entry, or that eventually $y \xrightarrow{j} x$ happens.

- If $x \xrightarrow{j} u_y$ happens, and if $t_y < t_x$, then x knows y from u_y's reply (which is before time t^e_x). If $t_y > t_x$, then similar to the above argument (in the case that $u_y \xrightarrow{j} x$ happens), if at time t_y, x is already an S-node, then $y \xrightarrow{j} x$ will happen; if $t_x > t_y$, either $y \xrightarrow{j} x$ happens, or x has stored a node in $(j-1+k, l)$-entry by the time it receives all the replies from node in contact-chain(x', x), where $x' = N_{u_y}(j-1+k, l_j)$ and $x' \neq x$. ■

Proposition A.7 *Suppose $x \in C_{l_{j+1}\ldots l_1\cdot\omega}$ and $x \notin C_{l_j\ldots l_1\cdot\omega}$, $1 \leq j \leq d-k-1$, $l_1,\ldots,l_{j+1} \in [b]$ (or suppose $x \in C_{l_1\cdot\omega}$). If $W_{l\cdot l_i\ldots l_1\cdot\omega} \neq \emptyset$, $1 \leq i \leq j$, $l \in [b]$ and $l \neq l_{i+1}$ (or if $W_{l\cdot\omega} \neq \emptyset$, $l \in [b]$ and $l \neq l_1$) then there exists a node y, $y \in W_{l\cdot l_i\ldots l_1\cdot\omega}$, such that $N_x(k+i, l) = y$ by time t^e (or there exists a node y, $y \in W_{l\cdot\omega}$, such that $N_x(k, l) = y$ by time t^e).*

Proof: By Proposition 5.1, if $W_{l\cdot l_i\ldots l_1\cdot\omega} \neq \emptyset$, then $C_{l\cdot l_i\ldots l_1\cdot\omega} \neq \emptyset$. Observe that since $x \notin C_{l_{j-1}\ldots l_1\cdot\omega}$, x can not be in any C-set $C_{l_{j'}\ldots l_1\cdot\omega}$, $j' < j-1$, either. (Recall that we have defined $l_i\ldots l_1$ to be empty if $i = 0$.)

We know that x initially is given node g_0, $g_0 \in V$, to start joining. Consider *contact-chain(x, g_0)*, $(g_0, g_1, \ldots, g_k, \ldots, g_{k+f}, u_1, \ldots, u_h, x)$, where x requests level-i' neighbors from $g_{i'}$, $0 \leq i' \leq k+f$ (or $0 \leq i' \leq k+f-1$, in the case where after requesting level-$k+f-1$ neighbors from g_{k+f-1}, x finds that g_{k+f} is still a T-node), sends a *JW* to g_{k+f}, finds that u_1 is stored by g_{k+f} in the corresponding entry, and sends another *JW* to u_1 and so on, until it is stored by u_h and receives a positive reply from u_h. Note that nodes g_0 to g_k are in V, and in particular, $g_k \in V_\omega$.

With nodes g_k to x, we can create a *suffix-chain(g_k, x)* as follows (intuitively, if $u_{h'+1}$, $1 \leq h' \leq h-1$, shares more than one digits with x than $u_{h'}$ does, then we insert several $u_{h'}$ between them, so that at the end, we get $j+1$ nodes in $suffix-chain(g_k, x)$, where $j = 1$ is the same number with that of C-sets from root to $C_{l_{j+1}\ldots l_1\cdot\omega}$):

- Put nodes g_k to g_{k+f} to *suffix-chain(g_k, x)* and preserve the order of the nodes.
- Initially, let $h' = 1$ and $f' = 1$, then repeat the following process until $f' = j$: If $|csuf(u_{h'}.ID, g_{k+f'}.ID)| = k + f'$, then let $g_{k+f'+1} = u_{h'}$ and increment both f' and h'; Otherwise, let $g_{k+f'+1} = g_{k+f'}$ and only increment f'.

Then, we get *suffix-chain(g_k, x)=($g_k, g_{k+1}, \ldots, g_{k+j}$)*.[12] Moreover, for each $g_{k+j'}$ in the chain, either $x \xrightarrow{c} g_{k+j'}$ or $x \xrightarrow{j} g_{k+j'}$ happens. Since $g_k \in V_\omega$, by the definition of $cset(V, W)$, $g_{k+1} \in C_{l_1\cdot\omega}$, $g_{k+2} \in C_{l_2 l_1\cdot\omega}$ and so on. In general, $g_{k+j'} \in C_{l_{j'}\ldots l_1\cdot\omega}$, $1 \leq j' \leq j-1$. Hence, $g_{k+i} \in C_{l_i\ldots l_1\cdot\omega}$.

By Proposition A.6, there exists a node y, $y \in C_{l\cdot l_{i-1}\ldots l_1\cdot\omega}$, such that $N_{g_{k+i}}(k+i, l) = y$ by time t^e. By Corollary A.5, either that g_{k+i} sets $N_{g_{k+i}}(k+i, l) = y$ before time $t^e_{g_{k+i}}$, or that there exists a node z, $z \in C_{l\cdot l_{i-1}\ldots l_1\cdot\omega}$, such that $z \xrightarrow{j} g_{k+i}$ eventually happens. If g_{k+i} sets $N_{g_{k+i}}(k+i, l) = y$ before $t^e_{g_{k+i}}$, then no matter $x \xrightarrow{c} g_{k+i}$ or $x \xrightarrow{j} g_{k+i}$ happens, the time g_{k+i} replies to x is no earlier than $t^e_{g_{k+i}}$. Hence, x knows y from g_{k+i}'s reply and will set $N_x(i+k-1, l) = y$ if it has not filled that entry. In the second case, by Proposition A.3, there must exist a node v, $v \in W_{l\cdot l_{i-1}\ldots l_1\cdot\omega}$, such that $N_x(i+k-1, l) = v$ at time t^e (because $z \xrightarrow{j} g_{k+i}$ happens, $x \xrightarrow{j} g_{k+i}$ or $x \xrightarrow{c} g_{k+i}$ happens, and $z.noti\_level \leq k+i$).

Similarly, we can prove that if $x \in C_{l_1\cdot\omega}$ and $W_{l\cdot\omega} \neq \emptyset$, $l \in [b]$ and $l \neq l_1$, then there exists a node y, $y \in W_{l\cdot\omega}$, such that $N_x(k, l) = y$ by time t^e. ■

Proposition 5.3 *For any node x, $x \in W$, if $W_{l\cdot l_i\ldots l_1\cdot\omega} \neq \emptyset$, where $l \in [b]$ and $l_i\ldots l_1\cdot\omega$ is a suffix of $x.ID$, $1 \leq i < d-k$, then $N_x(i+k, l) = y$ by time t^e, $y \in W_{l\cdot l_i\ldots l_1\cdot\omega}$; if $W_{l\cdot\omega} \neq \emptyset$, $l \in [b]$, then $N_x(k, l) = y$, $y \in W_{l\cdot\omega}$.*

Proof: Suppose $C_{l_j\ldots l_1\cdot\omega}$ is the first C-set x belongs to, then x also belongs to $C_{l_{j'}\ldots l_1\cdot\omega}$, $j \leq j' \geq d-k$. If $1 \leq i < j$, by Proposition A.7, if $W_{l\cdot l_i\ldots l_1\cdot\omega} \neq \emptyset$, then there exists a node y, such that $N_x(i+k, l) = y$ by time t^e. If $j \leq i \leq d-k-1$, then by part(b) of Proposition A.6, there exists a node y, such that $N_x(i+k, l) = y$ by time t^e. If $W_{l\cdot\omega} \neq \emptyset$, $l \in [b]$, then by ■

Proposition A.8 *For each node x, $x \in V \cup W$, if $(V \cup W)_{j\cdot x[i-1]\ldots x[0]} \neq \emptyset$, $i \in [d]$, $j \in [b]$, then there exists a node y, $y \in (V \cup W)_{j\cdot x[i-1]\ldots x[0]}$, such that by time t^e, $N_x(i, j) = y$.*

Proof: If $j = x[i]$, then $N_x(i, j) = x$ for all i. In what follows, we assume $j \neq x[i]$, $i \in [b]$. First, pick any node x, $x \in W$.

[12] For example, if node 00261 has requested level-0 neighbors from 30701, level-1 from 30701, level-2 from 10261, and then notifies 10261 and is stored by 10261 as $N_{10261}(4, 2)$, then *suffix-chain(30701,00261)* is as follows:(30701, 30701, 10261, 10261, 00261).

- If $0 \leq i < k$, then by Corollary A.1, if $V_{j\cdot x[i-1]...x[0]} \neq \emptyset$, then by time t^e there exists a node $y, y \in (V \cup W)_{j\cdot x[i-1]...x[0]}$, such that $N_x(i,j) = y$.
- If $i = k$ and $V_{j\cdot x[i-1]...x[0]} \neq \emptyset$, then again by Corollary A.1, there exists a node y, $y \in (V \cup W)_{j\cdot x[i-1]...x[0]}$, such that $N_x(i,j) = y$ by time t^e. If $i = k$, $(V \cup W)_{j\cdot x[i-1]...x[0]} \neq \emptyset$, however, $V_{j\cdot x[i-1]...x[0]} = \emptyset$, then $W_{j\cdot x[i-1]...x[0]} \neq \emptyset$. By Proposition 5.3, there exists a node y, $y \in W_{j\cdot x[i-1]...x[0]}$, such that $N_x(i,j) = y$ by time t^e.
- If $k < i \leq d-1$, then $(V \cup W)_{j\cdot x[i-1]...x[0]} = W_{j\cdot x[i-1]...x[0]}$. By Proposition 5.3, if $W_{j\cdot x[i-1]...x[0]} \neq \emptyset$, then there exists a node y, $y \in W_{j\cdot x[i-1]...x[0]}$, such that $N_x(i,j) = y$ by time t^e.

Second, consider nodes in V. Pick x, $x \in V$. If $(V \cup W)_{j\cdot x[i-1]...x[0]} \neq \emptyset$ and $V_{j\cdot x[i-1]...x[0]} \neq \emptyset$, then given that $\mathcal{N}(V)$ is consistent, there exists a node $y, y \in V_{j\cdot x[i-1]...x[0]}$ such that $N_x(i,j) = y$. If $(V \cup W)_{j\cdot x[i-1]...x[0]} \neq \emptyset$ and $V_{j\cdot x[i-1]...x[0]} = \emptyset$, then it must be that $x \in V_\omega$ and $x[i-1]...x[0] = \omega$. By Proposition 5.2, there exists a node y, $y \in W_{j\cdot x[i-1]...x[0]}$ such that $N_x(i,j) = y$. ■

Proposition 5.4 *For any two nodes x and y, $x \in V \cup W$, $y \in V \cup W$, $\langle x \rightarrow y \rangle_d$ by time t^e.*

Proof: By Lemma 3.1 and Proposition A.8, the proposition holds. ■

So far, the propositions we have proved are all about a set of joining nodes that belong to the same C-set tree. Next, we consider the case where the joining nodes belong to different C-set trees.

Proposition A.9 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, join a consistent network $\langle V, \mathcal{N}(V) \rangle$ concurrently. Let $G(V_{\omega_1}) = \{x, x \in W, V_x^{Notify} = V_{\omega_1}\}$, $G(V_{\omega_2}) = \{y, y \in W, V_y^{Notify} = V_{\omega_2}\}$, where $V_{\omega_1} \cap V_{\omega_2} = \emptyset$. For any node x, $x \in G(V_{\omega_1})$, if $(V \cup G(V_{\omega_1}) \cup G(V_{\omega_2}))_{j\cdot x[i-1]...x[0]} \neq \emptyset$, then there exists a node y, $y \in V \cup G(V_{\omega_1}) \cup G(V_{\omega_2})$, such that $N_x(i,j) = y$ by time t^e.*

Proof: If $j = x[i]$, then $N_x(i,j) = x$ for all i. In what follows, we assume $j \neq x[i]$, $i \in [b]$. Since $V_{\omega_1} \cap V_{\omega_2} = \emptyset$, neither ω_1 nor ω_2 is a suffix of the other. Consider any node x, $x \in G(V_{\omega_1})$.

- For $0 \leq i < k_1$, if $(V \cup G(V_{\omega_1}) \cup G(V_{\omega_2}))_{j\cdot x[i-1]...x[0]} \neq \emptyset$, then it must be $V_{j\cdot x[i-1]...x[0]} \neq \emptyset$. Otherwise, since $G(V_{\omega_1})_{j\cdot x[i-1]...x[0]} = \emptyset$ ($j \neq x[i]$, hence $j \cdot x[i-1]...x[0]$ is not a suffix of ω_1), it has to be $G(V_{\omega_2})_{j\cdot x[i-1]...x[0]} \neq \emptyset$. However, $V_{j\cdot x[i-1]...x[0]} = \emptyset$ and $G(V_{\omega_2})_{j\cdot x[i-1]...x[0]} \neq \emptyset$ indicates ω_2 is a suffix of $x[i-1]...x[0]$, thus a suffix of ω_1. A contradiction. Hence, $V_{j\cdot x[i-1]...x[0]} \neq \emptyset$. Then, by Corollary A.1, by time t^e, $N_x(i,j) = z$, $z \in V_{j\cdot x[i-1]...x[0]}$.
- If $i = k_1$ and $V_{j\cdot x[i-1]...x[0]} \neq \emptyset$, then by Corollary A.1, by time t^e, $N_x(i,j) = z$, $z \in V_{j\cdot x[i-1]...x[0]}$.
- If $k_1 \leq i \leq d-1$ and $(V \cup G(V_{\omega_1}) \cup G(V_{\omega_2}))_{j\cdot x[i-1]...x[0]} \neq \emptyset$, however, $V_{j\cdot x[i-1]...x[0]} = \emptyset$, then $(V \cup G(V_{\omega_2}))_{j\cdot x[i-1]...x[0]} = \emptyset$ (by Corollary A.3), hence, it could only be $G(V_{\omega_1})_{j\cdot x[i-1]...x[0]} \neq \emptyset$. Then, by Proposition 5.3, by time t^e, $N_x(i,j) = z$, $z \in G(V_{\omega_1})_{j\cdot x[i-1]...x[0]}$.

■

Proposition 5.5 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, join a consistent network $\langle V, \mathcal{N}(V) \rangle$ concurrently. Let $G(V_{\omega_1}) = \{x, x \in W, V_x^{Notify} = V_{\omega_1}\}$, $G(V_{\omega_2}) = \{y, y \in W, V_y^{Notify} = V_{\omega_2}\}$, $\omega_1 \neq \omega_2$.[13] Then by time t^e,*

- *$\forall x, \forall y$, $x \in G(V_{\omega_1})$, $y \in G(V_{\omega_2})$, $\langle x \rightarrow y \rangle_d$.*

Proof: If neither ω_1 nor ω_2 is a suffix of the other, then $V_{\omega_1} \cap V_{\omega_2} = \emptyset$ (by Lemma A.1). Hence, for any x, $x \in G(V_{\omega_1})$, and any y, $y \in G(V_{\omega_2})$, $\langle x \rightarrow y \rangle_d$ (by Lemma 5.3). [14]

Next, suppose one of ω_1 and ω_2 is a suffix of the other. By Lemma 3.1, to prove $\langle x \rightarrow y \rangle_d$, it is sufficient to prove that $\mathcal{N}(V \cup G(V_{\omega_1}) \cup G(V_{\omega_2}))$ is consistent by time t^e. First, consider nodes in V. Clearly, to maintain the consistency of the network, only nodes in V_{ω_1} and nodes in V_{ω_2} needs to update some of their table entries. By Proposition 5.2, for each $u, u \in V_{\omega_1}$, if $G(V_{\omega_1})_{j\cdot\omega_1} \neq \emptyset$, then there exists a node v, $v \in G(V_{\omega_1})_{j\cdot\omega_1}$, such that $N_u(k_1, j) = v$. Similarly, for each u, $u \in V_{\omega_2}$, if $G(V_{\omega_2})_{j\cdot\omega_2} \neq \emptyset$, then there exists a node v, $v \in G(V_{\omega_2})_{j\cdot\omega_2}$, such that $N_u(k_2, j) = v$.

Without loss of generality, suppose ω_2 is a proper suffix of ω_1. Let $k_1 = |\omega_1|$ and $k_2 = |\omega_2|$. Then it follows that $k_2 < k_1$ and $V_{\omega_1} \subset V_{\omega_2}$.

For any node x, $x \in G(V_{\omega_1})$, if $j = x[i]$, $i \in [b]$, then $N_x(i,j) = x$; if $j \neq x[i]$:

- If $0 \leq i \leq k_2$ and $V_{j\cdot x[i-1]...x[0]} \neq \emptyset$, then by Corollary A.1, by time t^e, $N_x(i,j) = z$, $z \in V_{j\cdot x[i-1]...x[0]}$. (Note that for $0 \leq i < k_2$, if $V_{j\cdot x[i-1]...x[0]} = \emptyset$, then $(V \cup G(V_{\omega_1}) \cup G(V_{\omega_2}))_{j\cdot x[i-1]...x[0]} = \emptyset$.)
- If $i = k_2$ and $V_{j\cdot x[i-1]...x[0]} = \emptyset$, however, $(V \cup G(V_{\omega_1}) \cup G(V_{\omega_2}))_{j\cdot x[i-1]...x[0]} \neq \emptyset$, then it must be that $G(V_{\omega_1})_{j\cdot x[i-1]...x[0]} = \emptyset$ (because $j \neq x[i]$), however, $G(V_{\omega_2})_{j\cdot x[i-1]...x[0]} \neq \emptyset$. Suppose g_0 is the node

[13] For example, if $V = \{30701, 11361\}$ and $W = \{00261, 30451, 74261\}$, then $G(V_{61}) = \{00261, 74261\}$ and $G(V_1) = \{30451\}$, where $\omega_1 = 61$ and $\omega_2 = 1$. Intuitively, this means that if only 00261 (or 74261) joins the network, then it needs to notify all the nodes in V_{61}, and if only 30451 joins the network, then it needs to notify all the nodes in V_1.

[14] Even if nodes in W join dependently, it is possible that there exist two nodes x and y, $x \in W$, $y \in W$, such that $V_x^{Notify} \cap V_y^{Notify} = \emptyset$. In such a case, there exists a node z, $z \in W$, such that $V_x^{Notify} \cap V_z^{Notify} \neq \emptyset$ and $V_y^{Notify} \cap V_z^{Notify} \neq \emptyset$, By Definition 3.6 the joins of x, y and z are dependent.

x is given to start joining. Then consider the nodes x sends *CP* to, the first k_1+1 nodes are g_0, g_1, ..., g_{k_2}, ..., g_{k_1}, where $g_{k_2} \in V_{\omega_2}$ and $g_{k_1} \in V_{\omega_1}$ (g_k, $0 \le k \le k_1$, shares k rightmost digits with x, and $x[k_1]...x[0]$ is ω_2 according to our assumption). Hence, $x \overset{c}{\rightarrow} g_{k_2}$ happens. Moreover, by Proposition 5.2, there exists a node z, $z \in G(V_{\omega_2})_{j \cdot x[k_2-1]...x[0]}$, such that $z \overset{j}{\rightarrow} g_{k_2}$ happens. Hence, by Proposition A.3, there exists a node z', $z' \in G(V_{\omega_2})_{j \cdot x[k_2-1]...x[0]}$, such that $N_x(k_2, j) = z'$ by time t^e.

- If $k_2 < i \le d-1$ and $(V \cup G(V_{\omega_1}) \cup G(V_{\omega_2})_{j \cdot x[i-1]...x[0]} \neq \emptyset$, then it must be that $G(V_{\omega_1})_{j \cdot x[i-1]...x[0]} \neq \emptyset$. By Proposition 5.3, by time t^e, $N_x(i, j) = z$, $z \in G(V_{\omega_1})_{j \cdot x[i-1]...x[0]}$.

For any node y, $y \in G(V_{\omega_2})$, if $j = y[i]$, $i \in [b]$, then $N_y(i, j) = y$; if $j \neq y[i]$:

- If $0 \le i \le k_2$ and $V_{j \cdot y[i-1]...y[0]} \neq \emptyset$, then by Corollary A.1, by time t^e, $N_y(i, j) = z$, $z \in V_{j \cdot y[i-1]...y[0]}$. (Again, for $0 \le i < k_2$, if $V_{j \cdot y[i-1]...y[0]} \neq \emptyset$, then $(V \cup G(V_{\omega_1}) \cup G(V_{\omega_2}))_{j \cdot y[i-1]...y[0]} = \emptyset$.)
- If $i = k_2$, $(V \cup G(V_{\omega_1}) \cup G(V_{\omega_2}))_{j \cdot y[i-1]...y[0]} \neq \emptyset$, however, $V_{j \cdot y[i-1]...y[0]} = \emptyset$, then it must be $G(V_{\omega_1})_{j \cdot y[i-1]...y[0]} = \emptyset$ and $G(V_{\omega_2})_{j \cdot y[i-1]...y[0]} \neq \emptyset$. Otherwise, if $G(V_{\omega_1})_{j \cdot y[i-1]...y[0]} \neq \emptyset$, then there exists x, $x \in G(V_{\omega_1})$, such that $x[k_2] = j$. By assuming ω_2 is a proper suffix of ω_1, we have $x[k_2]...x[0] = j \cdot y[k_2-1]...y[0]$. However, $V_x^{Notify} = V_{x[k_1-1]...x[0]}$, $k_1 > k_2$ by the assumption, indicates that $V_{x[k_2]...x[0]} \neq \emptyset$, i.e., $V_{j \cdot y[i-1]...y[0]} \neq \emptyset$. A contradiction. Having proved that in this case, $G(V_{\omega_2})_{j \cdot y[i-1]...y[0]} \neq \emptyset$, by Proposition 5.3, we conclude that by time t^e, $N_y(i, j) = z$, $z \in G(V_{\omega_2})_{j \cdot y[i-1]...y[0]}$.
- If $k_2 < i \le d-1$ and $(V \cup G(V_{\omega_1}) \cup G(V_{\omega_2}))_{j \cdot y[i-1]...y[0]} \neq \emptyset$, then it must be $G(V_{\omega_1})_{j \cdot y[i-1]...y[0]} = \emptyset$ and $G(V_{\omega_2})_{j \cdot y[i-1]...y[0]} \neq \emptyset$. By Proposition 5.3, by time t^e, $N_y(i, j) = z$, $z \in G(V_{\omega_2})_{j \cdot y[i-1]...y[0]}$.

■

Lemma 5.4 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \ge 2$, join a consistent network $\langle V, \mathcal{N}(V)\rangle$ concurrently. If the joins are dependent, then at time t^e, $\langle V \cup W, \mathcal{N}(V \cup W)\rangle$ is consistent.*

Proof: First, separate nodes in W into groups $\{G(V_{\omega_i})$, $1 \le i \le h\}$, where $\omega_i \neq \omega_j$ if $i \neq j$, such that for any node x in W, $x \in G(V_{\omega_i})$ if and only if $V_x^{Notify} = V_{\omega_i}$, $1 \le i \le h$. Consider any two nodes x and y. Then, by time t^e,

- If x and y are both in V, or $x \in V$ and $y \in G(V_{\omega_i})$, $1 \le i \le h$, or both x and y belong to $G(V_{\omega_i})$, then by Proposition 5.4, x and y can reach each other.
- If $x \in G(V_{\omega_i})$, $y \in G(V_{\omega_j})$, $1 \le i \le h$, $1 \le j \le h$, $i \neq j$, then by Proposition 5.5, x and y can reach each other.

Therefore, any two nodes in $V \cup W$ can reach each other by time t^e. By Lemma 3.1, $\langle V \cup W, \mathcal{N}(V \cup W)\rangle$ is consistent at time t^e. ■

Lemma 5.5 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \ge 2$, join a consistent network $\langle V, \mathcal{N}(V)\rangle$ concurrently. Then at time t^e, $\langle V \cup W, \mathcal{N}(V \cup W)\rangle$ is consistent.*

Proof: First, separate nodes in W into groups, such that joins of nodes in the same group are dependent and joins of nodes in different groups are mutually independent, as follows (initially, let $i = 1$):

- For each node y, $y \in W - \bigcup_{j=1}^{i} G_j$, if there exists a node x, $x \in G_i$, such that $(V_y^{Notify} \cap V_x^{Notify} \neq \emptyset)$ or $(\exists u, u \in W - \bigcup_{j=1}^{i-1} G_j$, $(V_y^{Notify} \subset V_u^{Notify}) \wedge (V_x^{Notify} \subset V_u^{Notify}))$, put y in G_i;
- Pick any node x', $x' \in W - \bigcup_{j=1}^{i} G_j$, put x' in G_{i+1}, increment i and repeat these two steps until there is no node left.[15]

Then, we get groups $\{G_i, 1 \le i \le l\}$. It can be checked that for any node x, $x \in G_i$, $1 \le i \le l$, $V_x^{Notify} \cap V_y^{Notify} = \emptyset$, $y \in G_j$, $1 \le j \le l$ and $i \neq j$.

Consider any two nodes x and y.

- If both x and y are in V, or $x \in V$, $y \in G_i$, or both x and y are in G_i, then by Lemma 5.4, x and y can reach each other by time t^e.
- If $x \in G_i$, $y \in G_j$, $i \neq j$, then $V_x^{Notify} \cap V_y^{Notify} = \emptyset$. By Proposition A.9, x and y can reach each other by time t^e.

Therefore, any two nodes in $V \cup W$ can reach each other by time t^e. By Lemma 3.1, $\langle V \cup W, \mathcal{N}(V \cup W)\rangle$ is consistent at time t^e. ■

Theorem 1 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \ge 1$, 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.*

Proof of Theorem 1: If $m = 1$, then by Lemma 5.1, the theorem holds.

[15] For example, suppose $V = \{72430, 10353, 62332, 13141, 31701\}$ and $W = \{23241, 00701, 47051, 47320\}$. First, let $G_1 = \{23241\}$. Nodes in $W - G_1$ are then checked one by one. Let y be the node that is being checked. (i) $G_1 = \{23241\}$, $y = 00701$. Then there exists a node x, $x = 23241$ ($x \in G_1$), and a node u, $u = 47051$ ($u \in W$), such that $V_y^{Notify} \in V_u^{Notify}$ and $V_x^{Notify} \in V_u^{Notify}$ ($V_y^{Notify} = V_{01}$, $V_x^{Notify} = V_{41}$, $V_u^{Notify} = V_1$). Hence, 00701 is included in G_1. (ii) $G_1 = \{23241, 00701\}$, $y = 47051$. Then there exists a node x, $x = 23241$ ($x \in G_1$), such that $V_y^{Notify} \cap V_x^{Notify} \neq \emptyset$. 47051 is also included in G_1. (iii) $G_1 = \{23241, 00701, 47051\}$, $y = 47230$. Neither of the condition mentioned above is satisfied. Thus, y is not included in G_1. (iv) Put 47230 in G_2, and there is no more node left. Eventually, nodes in W are separated into two groups, G_1 and G_2.

If $m \geq 2$, then according to their joining periods, nodes in W can be separated into several groups, $\{G_i, 1 \leq i \leq l\}$, such that nodes in the same group join concurrently and nodes in different groups join sequentially. Let the joining period of G_i be $[t^b_{G_i}, t^e_{G_i}]$, $1 \leq i \leq l$, where $t^b_{G_i} = \min(t^b_x, x \in G_i)$ and $t^e_{G_i} = \max(t^e_x, x \in G_i)$. We number the groups in such a way that $t^e_{G_i} < t^b_{G_{i+1}}$. Then, if $|G_1| \geq 2$, by Lemma 5.5, at time $t^e_{G_1}$, $\langle V \cup G_1, \mathcal{N}(V \cup G_1)\rangle$ is consistent; if $|G_1| = 1$, then by Lemma 5.1, $\langle V \cup G_1, \mathcal{N}(V \cup G_1)\rangle$ is consistent at time $t^e_{G_1}$. Similarly, by applying Lemma 5.5 (or Lemma 5.1) to G_2, ..., G_l, we conclude that evetually, at time t^e, $\langle V \cup W, \mathcal{N}(V \cup W)\rangle$ is consistent. ■

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.*

Proof: For each node x, $x \in W$, there are three phases before it becomes an S-node: c-phase, w-phase and n-phase. We then prove that x will proceed from c-phase to w-phase, from w-phase to n-phase, and evetually n-phase will end. Our proof is based upon the assumption of reliable message delivery and no node deletion during joins. Observe that among all the messages a joining node may send, *CP*, *JW*, *JN*, and *SN* need to be replied. The other types of messages do not require replies and thus do not prevent x from entering a new phase.

- First, consider a joining node, x, in c-phase. In this phase, x copies neighbors from other nodes to construct its own table by sending *CP* and receiving *CPRly*. The process of sending a request and receiving a reply can at most be d rounds, since there are only d levels in a table and at each round, i is incremented, where i indicates which level of neighbors are requested. Moreover, once a node receives a copy-request, it replies to x with no waiting. Therefore, x eventually proceeds to w-phase.

- Second, consider a joining node, x, in w-phase. In this phase, x sends out *JW*. Suppose x stops at level-k at the end of c-phase, then x can at most send out $d - k$ *JW* before it receives a positive *JWRly*, because each time it sends one more *JW*, the receiver shares at least one more digit with x than the previous receiver. (When it sends another *JW*, it has received the reply to its previous *JW*.) We next show that after sending a *JW*, x eventually receives a reply. If the receiver of a *JW*, y, is an S-node, then y replies with no waiting. If y is not yet an S-node, then it is a joining node in n-phase (or is about to enter n-phase)[16] and will wait until it becomes an S-node before replying to x. Thus, to complete the proof of this theorem, it suffices to show that any joining node in n-phase eventually becomes an S-node.

- Last, consider a joining node, z, in n-phase. There are two types of messages sent by z in this phase, *JN* and *SN* that need to be replied. z only sends *JN* to a subset of nodes in $V \cup W$ that share the rightmost i digits with itself, $i = z.noti\_level$, and each receiver of a *JN* replies to z with no waiting. Also, z only sends *SN* to a subset of nodes in W that share the rightmost $i + 1$ digits with it. Each *SN* is forwarded at most d time before a reply is sent to z, and each receiver of the message can reply to z or forward the message to another node with no waiting. Therefore, z eventually becomes an S-node. ■

A.2 Communication cost

The messages exchanged during a node's join can be categorized into the following sets:

1. *CP* and *CPRly*,
2. *JW* and *JWRly*,
3. *JN* and *JNRly*,
4. *SN* and *SNRly*,
5. *InSysNotiMsg*,
6. *RN* and *RNRly*

where messages in sets 1, 2 and 3 could be big in size, since they may include a copy of a neighbor table, while messages in sets 4, 5 and 6 are small in size. In Section 5.2, we have presented the number (or expected number) for messages in sets 1, 2 and 3 sent in a node's join process. In this section, we present proofs of Theorems 3, 4 and 5, and analyses of numbers of messages in sets 4, 5 and 6.

Theorem 3 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 1$, join a consistent network $\langle V, \mathcal{N}(V)\rangle$. Then, for any x, $x \in W$, the number of CpRstMsg and JoinWaitMsg sent by x is at most $d + 1$.*

Proof: Suppose at the end of c-phase, x has built its table up to level-i. Then, the number of *CP* sent by x in c-phase is at most $i + 1$ (recall that we number the levels from 0 to $d - 1$). In w-phase, x sends *JW* to u_1, u_2, and so on until a node u_j sends a positive reply to x. For each $u_{j'}$, $2 \leq j' \leq j$, $d - 1 \geq |csuf(x.ID, u_{j'}.ID)| \geq |csuf(x.ID, u_{j'-1}.ID) \geq i$. Hence, x can at most send $d - i$ *JW* during w-phase. Therefore, the total number of *CP* and *JW* sent by x is at most $d + 1$. ■

Theorem 4 *Suppose node x joins a consistent network $\langle V, \mathcal{N}(V)\rangle$, $|V| = n$. Then, the expected number of JoinNotiMsg sent by x is $\sum_{i=0}^{d-1} \frac{n}{b^i} P_i(n) - 1$, where $P_i(n)$*

[16] This is because that since x copies y from another node's neighbor table, y is already stored by that node, which can only happen at the end of y's w-phase or in y's n-phase.

is $\sum_{k=1}^{\min(n,B)} \frac{C(B,k)C(b^d-b^{d-i},n-k)}{C(b^d-1,n)}$ *for* $1 \le i < d-1$, *where* $B = (b-1)b^{d-1-i}$ *and* $C(B,k)$ *denotes number of* k*-combinations of* B *objects,* $P_0(n)$ *is* $\frac{C(b^d-b^{d-1},n)}{C(b^d-1,n)}$, *and* $P_{d-1}(n)$ *is* $1-\sum_{j=0}^{d-2} P_j(n)$.

Proof: Suppose $V_x^{Notify} = V_\omega$. Then x needs to notify all the nodes in V_ω. By Proposition A.5, there exists a node u_x, $u_x \in V_\omega$, such that u_x is the attaching-node of x ($A(x) = u_x$). Then, x sends a *JW* to u_x, however, x sends *JN* to any other node in V_ω (by Proposition A.1, for any node in V_ω other than u_x, x will send a *JN*). Hence, the number of *JN* x sends is $|V_\omega| - 1$. Let $Y = |\omega|$ and $Z = |V_\omega|$. We denote the probability that Y equals j given $|V| = n$ as $P_j(n)$, $j \in [d]$. Then, $P_j(n) = P(V_\omega \neq \emptyset \wedge V_{x[j]\cdot\omega} = \emptyset)$, i.e., $P_j(n) = P(V_{x[j-1]...x[0]} \neq \emptyset \wedge V_{x[j]...x[0]} = \emptyset)$. Then,

$$E(Z) = E(E(Z|Y)) = \sum_{i=0}^{d-1}(E(Z|Y=i))P_i(n) \qquad (1)$$

We derive $E(Z|Y=i))$ first, given $Y = i$, $V_\omega = V_{x[i-1]...x[0]}$. Since in a hypercube network, the node IDs are distributed randomly in the ID space $[b^d]$, the expect number of nodes in V whose IDs have suffix $x[i-1]...x[0]$ is $\frac{n}{b^i}$. Hence, $E(Z|Y=i) = \frac{n}{b^i}$.

Next, we compute $P_i(n)$, $i \in [d-1]$. In general, IDs of nodes in V can be drawn from $b^d - 1$ possible values. That is, for any y, $y \in V$, $y.ID$ could be any value from 0 to $b^d - 1$ except $x.ID$. If $i = 0$, then $V_{x[0]} = \emptyset$, i.e., there is no node in V whose ID has suffix $x[0]$. Then, IDs of nodes in V are drawn from $(b-1)b^{d-1}$ possible values (for any node y, $y \in V$, $y[0]$ could be any value in $[b]$ other than $x[0]$, and $y[j]$ could be any value in $[b]$, $1 \le j \le d-1$). Hence,

$$P_0(n) = \frac{\begin{pmatrix} b^d - b^{d-1} \\ n \end{pmatrix}}{\begin{pmatrix} b^d - 1 \\ n \end{pmatrix}}$$

If $1 \le i < d-1$, then $V_{x[i-1]...x[0]} \neq \emptyset$ and $V_{x[i]...x[0]} = \emptyset$. That is, at least there is one node in V with suffix $x[j-1]...x[0]$ but there is no node in V with suffix $x[i]...x[0]$. Then, for the n IDs of nodes in V k out of them are drawn from $(b-1)b^{d-1-i}$ possible values (for any node y, $y \in V_{x[i-1]...x[0]}$, $y[i]$ could be any value in $[b]$ except $x[i]$, and digits $y[d-1]$ to $y[i+1]$ could be any value in $[b]$) , $1 \le k \le \min(n,(b-1)b^{d-1-i})$, and the other $n-k$ IDs are drawn from $b^d - b^{d-i}$ possible values. Hence,

$$P_i(n) = \sum_{k=1}^{\min(n,B)} \frac{\begin{pmatrix} B \\ k \end{pmatrix}\begin{pmatrix} b^d - b^{d-i} \\ n-k \end{pmatrix}}{\begin{pmatrix} b^d - 1 \\ n \end{pmatrix}}$$

where $B = (b-1)b^{d-1-i}$.

Finally, for $i = d-1$, since each ID is unique, $x.ID$ is different than the ID of any node in V. Therefore, $V_{x[d-1]...x[0]} = \emptyset$ is always true, independent of whether $V_{x[d-2]...x[0]}$ is empty or not.

$$\begin{aligned} P_{d-1}(n) &= P(V_{x[d-1]...x[0]} = \emptyset \wedge V_{x[d-2]...x[0]} \neq \emptyset) \\ &= P(V_{x[d-1]...x[0]} = \emptyset)P(V_{x[d-2]...x[0]} \neq \emptyset) \\ &= P(V_{x[d-2]...x[0]} \neq \emptyset) \\ &= 1 - P(V_{x[d-2]...x[0]} = \emptyset) \\ &= 1 - P(V_{x[0]} = \emptyset \\ &\qquad \vee(V_{x[0]} \neq \emptyset \wedge V_{x[1]x[0]} = \emptyset) \vee ... \\ &\qquad \vee(V_{x[d-3]...x[0]} \neq \emptyset \wedge V_{x[d-2]...x[0]} = \emptyset)) \\ &= 1 - \sum_{i=0}^{d-2} P_i(n) \end{aligned}$$

Plug $P_i(n)$ into Equation 1, we get $E(Z)$. The expected number of *JN* x sends during its join is $E(Z) - 1$. ■

Theorem 5 *Suppose a set of nodes,* $W = \{x_1,...,x_m\}$, $m \ge 2$, *join a consistent network* $\langle V, \mathcal{N}(V)\rangle$. *Then for any node* x, $x \in W$, *an upper bound of the expected number of JoinNotiMsg sent by* x *is* $\sum_{i=0}^{d-1}(\frac{n+m}{b^i})P_i(n)$, *where* $n = |V|$, *and* $P_i(n)$ *is defined in Theorem 4.*

Proof: Consider any node x, $x \in W$. Let J be the number of *JN* sent by x when it joins with other nodes concurrently. Suppose $V_x^{Notify} = V_\omega$. Let $Y = |\omega|$ and $P_i(n)$ be the probability that Y equals i, $i \in [d]$, given $|V| = n$. No matter how many nodes join concurrently with x, $x.noti\_level \ge Y$. Moreover, x only sends *JN* to a subset of nodes whose IDs have suffix $x[k-1]...x[0]$, $k = x.noti\_level$. These nodes are a subset of nodes with suffix ω. Let $Z = |(V \cup W)_\omega|$. Hence, $J < Z$, which is true for every joining node. Therefore, $E(J) < E(Z)$. To compute $E(Z)$, we have

$$E(Z) = E(E(Z|Y)) = \sum_{i=0}^{d-1}(E(Z|Y=i))P_j(n)$$

where $E(Z|Y=i) = \frac{n+m}{b^i}$ and $P_j(n)$ is defined in Theorem 4. ■

Next, we present an upper bound of the expected number of messages in set 4, *SN* and *SNRly*. We say that an *SN* is initialized by x, if it is in the form of *SN*(x,y), where y could be any node other than x. Such a message is initially sent out by x to inform the receiver about the existence of y. It may be forwarded a few times before a reply is sent back to x. For example, x may send a *SN*(x,y) to u_1, u_1 forwards the same message to u_2, and u_2 sends a reply to x without further forwarding the message. In this example, there are 2 *SN*(x,y) and one *SNRly*(x,y) transmitted in the network.

Corollary A.6 *Suppose a set of nodes,* $W = \{x_1,...,x_m\}$, $m \ge 2$, *join a consistent network* $\langle V, \mathcal{N}(V)\rangle$. *Then for any* *node x, $x \in W$, an upper bound of the expected number of messages in the form of SN(x, y) or SNRly(x, y) sent by x, $y \neq x$, is $\sum_{i=0}^{d-1}(\frac{m}{b^{i+1}}(d-i-1))P_i(n)$, where $n = |V|$ and $P_i(n)$ is defined in Theorem 4.*

Proof: Consider any node x, $x \in W$. Suppose $V_x^{Notify} = V_\omega$, $i = |\omega|$, and $j = x.noti\_level$, then $j \geq i$. Let $X = \{y$, *SN(x, y)* is sent out by x during its join$\}$. Then, for a particular y, $y \in X$, *SN(x, y)* is only sent out by x once. Any y, $y \in X$, must share suffix $x[j]...x[0]$ with x. Since $V_{x[i]...x[0]} = \emptyset$ (by definition of V_x^{Notify}) and $j \geq i$, $V_{x[j]...x[0]} = \emptyset$. Hence, it could only be that $y \in W_{x[j]...x[0]}$, which is a subset of nodes in $W_{x[i]...x[0]}$.[17] The expected number of nodes in $W_{x[i]...x[0]}$ is $\frac{m}{b^{i+1}}$, thus, $E(|X|)$ can at most be $\frac{m}{b^{i+1}}$. For each *SN* sent out by x, it can be forwarded at most $d - j - 2$ times (which includes the first time that it is sent out by x), thus less than $d - i - 2$ times. This is because that the first receiver of the message shares at least $j + 2$ digits with y (both IDs of y and the first receiver must have suffix $y[j+1]...y[0]$, $y \in X$), the last receiver of the message shares at most $d-1$ digits with y, and each receiver along the path shares at least one more digit with y than the previous receiver does. Lastly, for each *SN(x, y* sent out by x, there is one corresponding reply, *SNRly(x, y)*, from the last receiver of the *SN(x, y)*.

Let $Y = |\omega|$, $Z = |W_{x[j]\cdot\omega}|$, (i.e., $Z = |W_{x[j]...x[0]}|$), and $P_i(n)$ be the probability that Y equals i given $|V| = n$, $i \in [d]$. Then,

$$E(Z) = E(E(Z|Y)) = \sum_{i=0}^{d-1}(E(Z|Y=i))P_j(n)$$

where $E(Z|Y = i) = \frac{m}{b^{i+1}}(d - i - 2 + 1)$ and $P_j(n)$ is defined in Theorem 4. ■

Figure 22 plots the upper bound of *SN* sent by a joining node, given n and m. In the simulations we conducted, however, we did not observe any *SN* that was sent out by a joining node.

To get the expected number of messages in set 5, *InSysNotiMsg*, suppose $V_x^{Notify} = V_\omega$. Then according to the join protocol, only a node with suffix ω may fill x into its neighbor table. (If a node's ID does not share any digits with ω, then clearly it will not choose x as a neighbor; if a node, y, shares a suffix ω' with x, $|\omega'| < |\omega|$, then by Corollary A.1, $N_y(k', x[k']) = z$, $z \in V$, $k' = |\omega'|$, thus x is not stored in y's table, either.) Let R denote the number of reverse-neighbors of x. At the end of its join, to each reverse-neighbor, x needs to send a *InSysNotiMsg*. Hence, the total number of messages in set 5 is R. Since the ID of a reverse-neighbor of x has suffix ω, the number of nodes in $V \cup W$ with suffix ω is an upper-bound of R. As defined in Theorem 5, this upper-bound is $\sum_{i=0}^{d-1}(\frac{n+m}{b^i})P_j(n)$, where $P_j(n)$ is defined in Theorem 4.

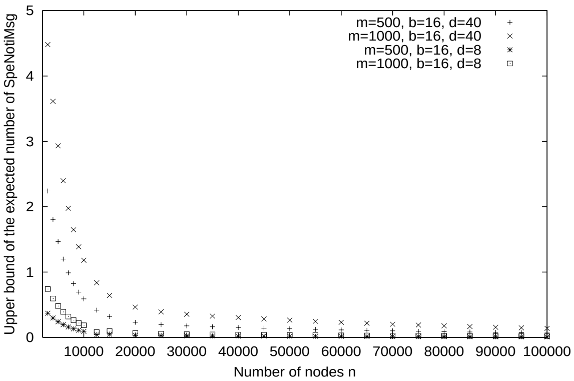


Figure 22. Theoretical upper bound of the expected number of SpeNotiMsg sent by a joining node

The number of messages in the last set, set 6, is $O(db)$, because x needs to inform each neighbor that x becomes a reverse-neighbor of it, by sending a *RN*. Some *RN* may be replied (when the status of the receiver kept by x is not consistent with the status of the receiver). Actually, some *RN* can be piggyback'ed with some other messages, such as *JWRly* and *JNRly*. Hence, the number of messages in set 6 that is sent by a joining node is at most $2db$.

[17] There are several extra conditions that constrain x from sending a *SN*, however, we ignore those conditions in deriving an upper bound.