Neighbor Table Construction and Update for Resilient Hypercube Routing in P2P Networks

Huaiyu Liu and Simon S. Lam

TR-07-31 July 12, 2007

***Abstract*—Several proposed peer-to-peer networks use hypercube routing for scalability. Consistency of neighbor tables in hypercube routing guarantees the existence of a path from any source node to any destination node. Such consistency, however, can be broken by node failures. To improve the robustness of hypercube routing, we first generalize the concept of *consistency* to K-*consistency*, for $K \geq 1$, which is shown to provide at least K disjoint paths for any source-destination pair with a probability close to 1. Our next objective is to design and specify a new join protocol together with a proof that it generates K-consistent neighbor tables for an arbitrary number of concurrent joins. We first present a conceptual foundation, called *C-set trees*, for protocol design and reasoning about K-consistency. We then present a detailed specification of a join protocol, and a rigorous proof of correctness for the join protocol. The crux of our proof is based upon induction on C-set trees. Both theoretical analysis and simulation results show that the join protocol is scalable to a large number of network nodes.**

***Keywords*—Peer-to-peer networks, consistency, K-consistency, hypercube routing, join protocol, protocol design and verification, C-set tree**

1 Introduction

Structured peer-to-peer (p2p) networks are being investigated as a platform for building large-scale distributed systems [11, 12, 13, 14, 16]. The primary function of these networks is object location, That is, mapping an object ID to a node in the network. For efficient routing, each node maintains $O(\log n)$ pointers to other nodes, called neighbor pointers, where n is the number of network nodes. To locate an object, the average number of application-level hops required is $O(\log n)$.[1] Each node stores neighbor pointers in a table, called its *neighbor table*. The neighbor tables constitute the routing infrastructure of a p2p network.

An important problem in p2p networks is the design and specification of protocols together with a proof that they construct and maintain *consistent* neighbor tables for network nodes that may join, leave, and fail concurrently. Of interest in this paper is the hypercube routing scheme used in several proposed p2p systems [11, 13, 16, 7]. Based on the hypercube routing scheme and additional distributed directory information, it is guaranteed to locate a copy of an object if it exists, and the expected cost of accessing is asymptotically optimal, given that the neighbor tables in the network are *consistent* (definition in Section 3) and *optimal* (that is, they store nearest neighbors) [11].

To implement the hypercube routing scheme in a dynamic, distributed environment, the following problems must be addressed:[2]

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*. (Hereafter, 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 to generate neighbor tables that are not only consistent, but also resilient to node failures. Our solution to the failure recovery problem for hypercube routing networks is presented in companion papers [5, 6].

Neighbor table consistency guarantees the existence of a path from any source node to any destination node in the network. Such consistency however can be broken by the failure of a single node. To provide resilience to node failures and facilitate the design of failure recovery protocols, we introduce a new concept, K-*consistency*, $K \geq 1$, which generalizes consistency (1-consistency is the same as consistency).[3] Informally, the neighbor tables of a network are K-consistent if and only if each table entry in every node stores $\min(K, H)$ neighbors, where H is the number of nodes in the network that have the "required suffix" (definition in Section 2) of the table entry. By providing redundancy in neighbor tables, K-consistency has the following advantages:

- K-consistency implies consistency and K-consistent neighbor tables provide "static resilience" [2]. More specifically, we show in Section 3 that a K-consistent network provides at least K disjoint paths from any node to any other node with probability approaching 1 as n increases (e.g., for $n = 300$ and $K = 3$, the probability is lower bounded by 0.99).
- K-consistency facilitates design of failure recovery protocol and supports rapid failure recovery. In the companion

Huaiyu Liu is with Communication Technology Lab, Intel Corporation, Hillsboro, OR 97124 USA (E-mail: huaiyu.liu@intel.com). Simon S. Lam is with the Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712-0233 USA (E-mail: lam@cs.utexas.edu, Fax: 1-512-471-8885). Research sponsored by NSF grants no. ANI-0319168 and CNS-0434515. This work is a major extension of our paper presented in [9].

[1]This is true except for CAN [12], in which routing takes $O(rn^{1/r})$ hops and r is the number of dimensions used in the system.

[2]For simplicity, we will say *network* instead of *hypercube routing network* and *table* instead of *neighbor table* whenever there is no ambiguity.

[3]In [9], we addressed 1-consistency and designed a join protocol that generates consistent neighbor tables. The major extension in this paper is generalization of 1-consistency to K-consistency.

papers [5, 6], we presented a failure recovery protocol that only uses local information, and integrated it with our join protocol presented in this paper. Through extensive simulation experiments, we found that for $K \geq 2$, all "recoverable holes" in neighbor tables due to failed nodes were repaired by the failure recovery protocol in *every* experiment. It was also shown in [5, 6] that the integrated protocols are able to maintain consistent neighbor tables under continuous and frequent node joins and leaves (churn).

- K-consistency benefits neighbor table optimization. In [10], we found that with the same set of optimization heuristics, a larger K value results in neighbor tables that provide shorter routes.

To design a join protocol that generates K-consistent neighbor tables for an arbitrary number of concurrent joins, a major difficulty is as follows. For every table entry in a joining node's table, the node needs to discover $\min(H, K)$ neighbors without any global knowledge, where H is the total number of nodes that have the required suffix of the entry and H could be any value equal to or greater than 0. (One approach to discover enough neighbors for an entry is through broadcasting, which is obviously not scalable.) To solve this problem, we first present a conceptual foundation, called *C-set trees*, for protocol design and reasoning about K-consistency. Second, based on the observation that in a K-consistent network, it is possible for a node to store the same neighbor at multiple levels in its neighbor table, we introduce a concept called *attach level*. It is a constraint on the lowest level that a joining node can be stored in a table and is important in the protocol for correctness.

In addition to the join protocol design, we also construct a rigorous proof of correctness for the join protocol (assuming reliable message delivery and no node failure or leave). The crux of our proof is based upon induction on C-set trees.

Contributions of this paper are the following:

- We define *K-consistency* for the hypercube routing scheme and demonstrate benefits of K-consistency via both theoretical analysis and simulation experiments.
- We analyze the goal of a join protocol for the hypercube routing scheme and present the detailed specification of a new join protocol. The join protocol can also be used for network initialization.
- We present a conceptual foundation, *C-set trees*, for protocol design and reasoning about K-consistency. By induction on C-set trees, we present a rigorous proof that the join protocol generates K-consistent neighbor tables for *an arbitrary number of concurrent joins*.
- We analyze communication costs of the join protocol as a function of K as well as of network size, and show that the protocol is scalable to a large number of network nodes.

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 [1, 3, 10] for methods of exploiting node proximity and optimizing neighbor tables.

The rest of this paper is organized as follows. In Section 2, we briefly review the hypercube routing scheme. In Section 3, we present our definition of K-consistency, and discuss advantages of K-consistency through both theoretical analysis and simulation experiments. In Section 4, our conceptual foundation for protocol design is presented and illustrated through examples. In Section 5, a detailed specification of our join protocol is presented. In Section 6, we present an outline of our correctness proof for the join protocol and analyze protocol performance. (Detailed proofs are presented in the Appendix.) Lastly, we show how to use the join protocol for network initialization in Section 7, discuss related work in Section 8, and conclude in Section 9.

2 Background

In this section, we briefly review the hypercube routing scheme used in PRR [11], Pastry [13], and Tapestry [16]. Consider a set of nodes. Each node has a unique ID, which is a fixed-length random binary string. A node's ID is represented by d digits of base b, e.g., a 160-bit ID can be represented by 40 Hex digits ($d = 40$, $b = 16$). Hereafter, we will use $x.ID$ to denote the ID of node x, $x[i]$ the ith digit in $x.ID$, and $x[i-1]...x[0]$ a suffix of $x.ID$. We count digits in an ID from right to left, with the 0th digit being the *rightmost* digit. See Table 1 for notation used throughout this paper.[4] Also, we will use "network" instead of "hypercube routing network" for brevity.

Notation	*Definition*
$\langle V, \mathcal{N}(V)\rangle$	a hypercube network: V is the set of nodes in the network, $\mathcal{N}(V)$ is the set of neighbor tables
$[\ell]$	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
$x[i]$	the ith digit in $x.ID$
$x[i-1]...x[0]$	suffix of $x.ID$; denotes empty string if $i = 0$
$x.table$	the neighbor table of node x
$j \cdot \omega$	digit j concatenated with suffix ω
$\lvert\omega\rvert$	the number of digits in suffix ω (length of suffix ω)
$N_x(i,j)$	the set of nodes in (i,j)-entry of $x.table$, also referred as the (i,j)-*neighbors* of node x
$N_x(i,j).size$	the number of nodes in $N_x(i,j)$
$N_x(i,j).first$	the first node in $N_x(i,j)$
$csuf(\omega_1, \omega_2)$	the longest common suffix of ω_1 and ω_2
$V_{l_i \ldots l_0}$	a *suffix set* of V, which includes all of the nodes in V that has an ID with suffix $l_i...l_0$; denotes V if $l_i...l_0$ is the empty string
$\lvert V\rvert$	the number of nodes in set V

Table 1: Notation

Given a message with destination node ID, $z.ID$, the objective of each step in hypercube routing is to forward the message from its current node, say x, to a next node, say y, such that the suffix match between $y.ID$ and $z.ID$ is at least one digit longer than the match between $x.ID$ and $z.ID$.[5] If such a path exists, the destination is reached in $O(\log_b n)$ steps on the average and d steps in the worst case, where n is the number of network nodes. Figure 1 shows an example path for routing from source node 21233 to destination node 03231 ($b = 4, d = 5$). Note that the ID of each intermediate node in the path matches 03231 by at least one more suffix digit than its predecessor.

To implement hypercube routing, each node maintains a *neighbor table* that has d levels with b entries at each level.

[4]In our notation, we use $V_{l_i \ldots l_0}$ to denote a suffix set of V. Similarly, $W_{l_i \ldots l_0}$ is a suffix set of W and $(V \cup W)_{l_i \ldots l_0}$ is a suffix set of $V \cup W$. However, we reserve $C_{l_i \ldots l_0}$ to denote a C-set, as defined in Section 4.

[5]In this paper, we follow PRR [11] and use suffix matching, whereas other systems use prefix matching. The choice is arbitrary and conceptually insignificant.

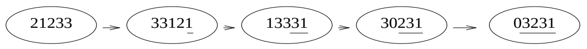
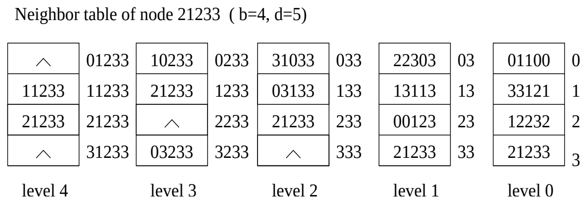


Figure 1: An example hypercube routing path

Each table entry stores link information to nodes whose IDs have the entry's required suffix, defined as follows. Consider the table in node x. The **required suffix** for entry j at level i, $j \in [b]$, $i \in [d]$, referred to as the (i, j)-entry of $x.table$, is $j \cdot x[i-1]...x[0]$. Any node whose ID has this required suffix is said to be a **qualified node** for the (i, j)-entry of $x.table$. Only qualified nodes for a table entry can be stored in the entry. Note that node x has the required suffix for each $(i, x[i])$-entry, $i \in [d]$, of its own table. For routing efficiency, we fill each node's table such that $N_x(i, x[i]).\mathit{first} = x$ for all $x \in V$, $i \in [d]$. Figure 2 shows an example neighbor table. The string to the right of each entry is the required suffix for that entry. An empty entry indicates that there does not exist a node in the network whose ID has the entry's required suffix.

Nodes stored in the (i, j)-entry of $x.table$ are called the *(i, j)-neighbors* of x, denoted by $N_x(i, j)$. Ideally, these neighbors are chosen from qualified nodes for the entry according to some proximity criterion [11]. Furthermore, node x is said to be a *reverse(i, j)-neighbor* of node y if y is an (i, j)-neighbor of x. Each node also keeps track of its reverse-neighbors. The link information for each neighbor stored in a table entry consists of the neighbor's ID and IP address. For clarity, IP addresses are not shown in Figure 2. Hereafter, we will use "neighbor" or "node" instead of "node's ID and IP address" whenever the meaning is clear from context.


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 2: An example neighbor table

3 K-consistent Networks

Constructing and maintaining consistent neighbor tables is an important design objective for structured p2p networks. We next present a rigorous definition of consistency and then introduce a stronger property, K-consistency, for the hypercube routing scheme.

Definition 3.1 *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 $V_{j\cdot x[i-1]...x[0]} \neq \emptyset$, $i \in [d]$, $j \in [b]$, then there exists a node y, $y \in V_{j\cdot x[i-1]...x[0]}$, such that $y \in N_x(i, j)$.
(b) If $V_{j\cdot x[i-1]...x[0]} = \emptyset$, $i \in [d]$, $j \in [b]$, then $N_x(i, j) = \emptyset$.

Part (a) in the above definition states that for each table entry, if there exists at least one node in the network that has the required suffix of the entry, then the entry must not be empty and it is filled with at least one node having the required suffix. Part (b) in the above definition states that if the network does not have any node with the required suffix of a particular table entry, then that table entry must be empty.

Definition 3.2 *Consider two nodes, x and y, in network $\langle V, \mathcal{N}(V)\rangle$. If there exists a neighbor sequence (a* **path***), $(u_0, ..., u_k)$, $k \leq d$, such that u_0 is x, u_k is y, and $u_{i+1} \in N_{u_i}(i, y[i])$, $i \in [k]$, then y is* **reachable** *from x, or x can* **reach** *y, in k hops.*[6]

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

Lemma 3.1 shows that neighbor table consistency guarantees the existence of a path from any source node to any destination node in the network. Such consistency however can be broken by the failure of a single node. To increase robustness and facilitate the design of failure recovery protocols, our original goal was to design a new join protocol that constructs a K-connected hypercube routing network, that is, a network in which neighbor tables provide at least K disjoint paths ($K > 1$) from any source node to any destination node. However, we quickly realized that for a network with a small number of nodes and some specific realization of node IDs, it is possible that a K-connected network does not exist. (Recall that node IDs are randomly generated.) This is because in hypercube routing, only "qualified" nodes whose IDs have the suffix required by a table entry can be stored in the table entry. Instead, we define a K-consistent (hypercube routing) network as follows:

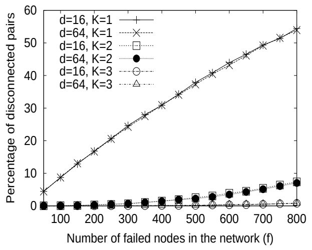
Definition 3.3 *Consider a network $\langle V, \mathcal{N}(V)\rangle$. The network, or $\mathcal{N}(V)$, satisfies K-***consistency***, $K \geq 1$, if for any node x, $x \in V$, each entry in its table satisfies the following conditions:*

(a) If $V_{j\cdot x[i-1]...x[0]} \neq \emptyset$, then $N_x(i, j).size = \min(K, |V_{j\cdot x[i-1]...x[0]}|)$, $i \in [d]$, $j \in [b]$, where $N_x(i, j) \subseteq V_{j\cdot x[i-1]...x[0]}$.
(b) If $V_{j\cdot x[i-1]...x[0]} = \emptyset$, $i \in [d]$, $j \in [b]$, then $N_x(i, j) = \emptyset$.

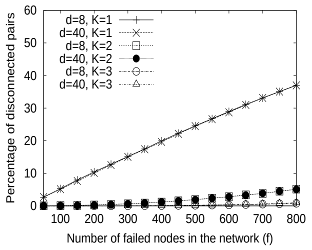
Definition 3.3 states that in a K-consistent network with n nodes, for every node in the network, each of its table entry is filled with K neighbors if there are K or more qualified nodes in the network for that entry; otherwise, all qualified nodes (if any) are stored in the entry. To study the resilience of K-consistent networks in the presence of failures, we first conducted simulation experiments as follows. We began by constructing a K-consistent network of n nodes following Definition 3.3, then randomly picked f nodes and let them fail. Next, we counted the number of disconnected source-destination pairs in the network. By a disconnected source-destination pair, (x, y), we mean that both x and y have not failed but x cannot reach y. Each simulation is identified by a combination of n, b, d, K and f values. For each combination, we ran five simulations and calculated the average value of the percentage of source-destination pairs that became disconnected.

Figure 3 shows some simulation results for percentages of disconnected source-destination pairs after node failures, for different number of failures in a network that initially had 4,000 nodes. First, note that the results are insensitive to the value of

[6]In this paper, k and K are used as different variables.



(a) b=4, n=4000



(b) b=16, n=4000

Figure 3: Percentage of disconnected source-destination pairs for different K values

d. In each plot, for each K value, the two curves for two different d values are almost the same. Second, when K is increased from 1 to 2, the percentage of disconnected pairs decreases dramatically. For $K = 3$, even after 20% of the nodes have failed, the number of disconnected source-destination pairs is less than 1% of all source-destination pairs. The results also show that increasing the value of b from 4 to 16 leads to a significant reduction in the percentage of disconnected source-destination pairs. This is because with a larger b, more neighbors are stored in a table (the number is proportional to $Kb\log_b n$). As expected, the simulation results show that with more neighbors stored in each entry, a network is more resilience in the presence of failures. (In fact, it is also easier for the network to recover from failures and maintain consistency of neighbor tables, as shown in [5].)

It is easy to see that K-consistency is a stronger property than consistency. In particular, a K-consistent network, $K \geq 1$, is a consistent network. In the balance of this paper, for each node x, we choose $N_x(i, x[i]).first$ to be x itself, $i \in [d]$, for efficient routing. Multiple neighbors stored in each table entry provide alternative paths from a source node to a destination node, and some of them are disjoint. More precisely, two paths from source node x to destination node y are *disjoint* if and only if any node in each path that is neither x nor y does not appear in the other path. Further, a set of paths from x to y are **disjoint** if and only if every pair of paths in the set are disjoint. For example, let a, b, and c denote nodes. Then the following paths are disjoint: $x \to y$, $x \to a \to y$, and $x \to b \to c \to y$.[7]

Theorem 1 *In a K-consistent network, $\langle V, \mathcal{N}(V)\rangle$, where $|V| = n$ and $n \geq K$, for any two nodes, x and y, $x \in V$, $y \in V$ and $x \neq y$, a lower bound of the probability that there exist at least K disjoint paths from x to y is $(1 - \frac{K-1}{n-1})\sum_{i=K}^{n} \frac{C(b^{d-1},i)C(b^d - b^{d-1}, n-i)}{C(b^d,n)}$, where $C(X,Y)$ is the number of Y-combinations of X objects.*

To prove Theorem 1, we first present two lemmas. Proofs of these lemmas are presented in Appendix A. Lemma 3.2 says that in a K-consistent network, if destination node y is not a neighbor stored in the table of node x, then at least K disjoint paths exist from x to y. However, if destination y is stored in $x.table$, then a tight lower bound of the number of disjoint paths from x to y depends upon whether y is stored in $N_x(0, x[0])$. Lemma 3.3 summarizes all the cases.

[7]Note that nodes here are user machines in a p2p network. Thus, it is possible for two disjoint paths in a K-consistent (hypercube routing) network to share a router in the underlying Internet. This would not be a concern since routers are generally much more resilient than user machines.

Lemma 3.2 *In a K-consistent network, $\langle V, \mathcal{N}(V)\rangle$, for any two nodes, x and y, $x \in V$, $y \in V$ and $x \neq y$, if $y \notin x.table$, then there exist at least K disjoint paths from x to y.*

Lemma 3.3 *In a K-consistent network, $\langle V, \mathcal{N}(V)\rangle$, for any two nodes, x and y, $x \in V$, $y \in V$ and $x \neq y$, if $y \notin N_x(0, x[0])$, then there exist at least $\min(K, |V_{y[0]}|)$ disjoint paths from x to y; if $y \in N_x(0, x[0])$, then there exist at least $\min(K, |V_{y[0]}|) - 1$ disjoint paths from x to y.*

Proof of Theorem 1: Let A be the event that there exist at least K disjoint paths from x to y, and B be the event that $y \notin N_x(0, x[0])$ (which includes $y \notin x.table$ and $y \in x.table \wedge y \notin N_x(0, x[0])$). Note that if $y \in N_x(0, x[0])$, then it must be that $y[0] = x[0]$. For any event X, let $P(X)$ denote the probability of X. We first derive $P(A \wedge B)$.

We know $P(A \wedge B) = P(A|B)P(B)$. $P(A|B)$ is the probability that there exist at least K disjoint paths from x to y, given $y \notin N_x(0, x[0])$. By Lemma 3.3, if $y \notin N_x(0, x[0])$, then there exist at least $\min(K, |V_{y[0]}|)$ disjoint paths from x to y. Thus, $P(A|B) = P(\min(K, |V_{y[0]}|) = K) = P(|V_{y[0]}| \geq K)$. $|V_{y[0]}| \geq K$ means that there exist at least K nodes in V with suffix $y[0]$.

$$P(A|B) = P(|V_{y[0]}| \geq K) = \sum_{i=K}^{n} \frac{C(b^{d-1}, i)C(b^d - b^{d-1}, n-i)}{C(b^d, n)}$$

To derive $P(B)$, let K' be the number of neighbors stored in $N_x(0, x[0])$ other than x itself. Since there are at most K nodes stored in $N_x(0, x[0])$ (by Definition 3.3) and one of them is x ($N_x(0, x[0]).first = x$), we have $K' \leq K - 1$.

$$P(B) = 1 - P(y \in N_x(0, x[0])) \geq 1 - \frac{K-1}{n-1}$$

Combining the above results, we have

$$\begin{aligned} P(A) &\geq P(A \wedge B) \\ &= P(A|B)P(B) \\ &= P(B)\sum_{i=K}^{n} \frac{C(b^{d-1}, i)C(b^d - b^{d-1}, n-i)}{C(b^d, n)} \\ &\geq (1 - \frac{K-1}{n-1})\sum_{i=K}^{n} \frac{C(b^{d-1}, i)C(b^d - b^{d-1}, n-i)}{C(b^d, n)} \end{aligned}$$

■

Figure 4(a) plots the lower bound of the probability that there exist at least K disjoint paths for every source-destination pair in a K-consistent network, where $b = 16$ and $d = 40$.[8] Observe that when n increases, the lower bound approaches 1. For example, the lower bound is higher than 0.99 for $n = 300$ and $K = 3$.

We complement the above analysis with simulation experiments. A set of simulations were conducted to evaluate the number of disjoint paths for each source-destination pair in K-consistent networks with different values of K, b, d and n. In each simulation, each node has a randomly generated ID, and the neighbor table of each node was constructed according to Definition 3.3, with $N_x(i, x[i]).first = x$ for all $x \in V$, $i \in [d]$. Then for each source-destination pair, the number of disjoint paths from source to destination was counted. For each combination of b, d, n and K values, we ran five simulations and

[8]$b = 16$ and $d = 40$ are commonly used values. Results for lower bounds of the probability with other values of b and d show the same trend.

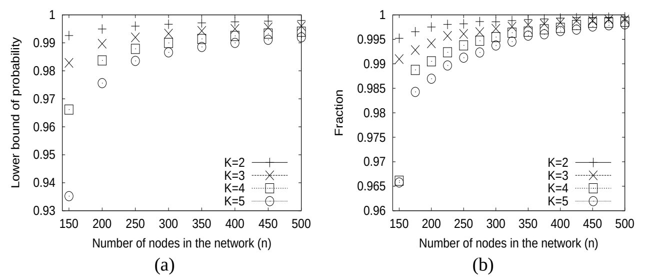


Figure 4: (a) Lower bound of the probability that there exist at least K disjoint paths for each source-destination pair, (b) Simulation results on the fraction of source-destination pairs with at least K disjoint paths. $b = 16$, $d = 40$

obtained the average value of the ratio of the number of source-destination pairs that have at least K disjoint paths to the total number of source-destination pairs. Figure 4(b) presents some of our simulation results. Observe that the results in Figure 4(a) are much closer to 1 than the corresponding lower bound results in Figure 4(a), as expected. For example, the fraction of source-destination pairs with at least K disjoint paths is greater than 0.996 for $n = 300$ and $K = 3$.

4 Conceptual Foundation

In this section, we first present definitions and assumptions to be used in our protocol design and proofs. Then we analyze the goals and tasks for a join protocol to produce K-consistent neighbor tables for the hypercube routing scheme, and present the concept of *C-set trees*.

4.1 Definitions and assumptions

Definition 4.1 *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 5). The period from t_x^b to t_x^e, denoted by $[t_x^b, t_x^e]$, is the* **joining period** *(or join duration) of x.*

Definition 4.2 *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 4.3 *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 4.4 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 1$, join a K-consistent network $\langle V, \mathcal{N}(V)\rangle$. For any node x, $x \in W$, if $|V_{x[k-1]...x[0]}| \geq K$ and $|V_{x[k]...x[0]}| < K$, $k \in [d]$, then $V_{x[k-1]...x[0]}$ is the* **notification set** *of x regarding V (or noti-set, in short).*

Intuitively, V_x^{Notify} is the set of nodes in V that need to update their neighbor tables to satisfy K-consistency conditions after the joins, if x were the only node that joins $\langle V, \mathcal{N}(V)\rangle$. For instance, suppose $x = 10261$ ($b = 8, d = 5$), and $V = \{13061, 31701, 11261, 10353\}$. If $K = 1$, then $V_x^{Notify} = V_{261} = \{11261\}$ ($V_{261} = \{11261\}$ and $V_{0261} = \emptyset$, thus $|V_{261}| \geq 1$ and and $|V_{0261}| < 1$, then by Definition 4.4, $V_x^{Notify} = V_{261}$); if $K = 2$, then $V_x^{Notify} = V_{61} = \{11261, 13061\}$; if $K = 3$, then $V_x^{Notify} = V_1 = \{11261, 13061, 31701\}$.

Definition 4.5 *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 4.6 *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 $V_y^{Notify} \subset V_u^{Notify}$.*

In designing the protocol for a node to join network $\langle V, \mathcal{N}(V)\rangle$, we make the following assumptions: (i) $V \neq \emptyset$ and $\langle V, \mathcal{N}(V)\rangle$ is a K-consistent network, (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 joins.

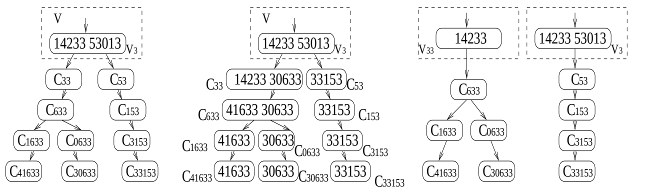
In a distributed p2p network, global knowledge is difficult (if not impossible) to get. Therefore, a node should utilize local information to construct or update neighbor tables. Under the assumption that there is no node deletion during joins, condition (b) in Definition 3.3 can be satisfied easily, since once a node has joined, it always exists in the network. Hence, given a K-consistent network, $\langle V, \mathcal{N}(V)\rangle$, and a set W of joining nodes, the goals of the join protocol are to construct neighbor tables for joining nodes and update tables of existing nodes such that eventually condition (a) in Definition 3.3 is satisfied in network $\langle V \cup W, \mathcal{N}(V \cup W)\rangle$. More specifically:

- **Goal 1:** For each node x, $x \in W$, and for each (i, j)-entry in $x.table$, $i \in [d]$ and $j \in [b]$, eventually $\min(K, H)$ nodes with suffix $j \cdot x[i-1]...x[0]$ are stored in the entry, where $H = |(V \cup W)_{j \cdot x[i-1]...x[0]}|$.
- **Goal 2:** For each node, y, $y \in V$, and for each (i, j)-entry in $y.table$, $i \in [d]$ and $j \in [b]$, if $N_y(i, j).size < K$ before the joins and $W_{j \cdot y[i-1]...y[0]} \neq \emptyset$, eventually the entry is updated and stores $\min(K, H)$ nodes with suffix $j \cdot y[i-1]...y[0]$, where $H = |(V \cup W)_{j \cdot y[i-1]...y[0]}|$.

4.2 C-set tree for K-consistency

If multiple nodes join a network sequentially, then the joins do not interfere with each other, because when a node joins, any node that joined ealier has already been integrated into the network. Also, if multiple nodes join a network concurrently and the joins are independent, then intuitively the joins do not interfere with each other either, because the sets of nodes that 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 nodes 30633 and 41633 join concurrently, each of them may think of itself as the only node with suffix 633 in the network. If handled incorrectly, views of the joining nodes may not converge eventually, which would result in inconsistent neighbor tables.

We first analyze the desirable results of multiple joins by using an example ($b = 8$, $d = 5$). Suppose a set of nodes, $W = \{30633,\ 41633,\ 33153\}$, join a K-consistent network $\langle V, \mathcal{N}(V)\rangle$, $V = \{02700,\ 14233,\ 53013,\ 62332,\ 72430\}$, and $K = 2$. Then by Definition 4.4, all nodes in W have the same noti-set, which is V_3.[9] Consider a joining node, say 33153. At the end of joins, for any y to reach 33153, $y \in V$, there should exist a neighbor sequence $(u_0, u_1, ..., u_5)$ such that u_0 is y, u_5 is 33153, and the IDs of u_1 to u_4 have suffix 3, 53, 153, and 3153, respectively. Since before the joins, $\langle V, \mathcal{N}(V)\rangle$ is K-consistent, y must have stored at least one neighbor with suffix 3, which is a node in V_3. Let the set of $(1, 5)$-neighbors of nodes in V_3 be C_{53}, the set of $(2, 1)$-neighbors of nodes in C_{53} be C_{153}, ..., and the set of $(4, 3)$-neighbors of nodes in C_{3153} be C_{33153}. We call these sets *C-sets* and the sequence of sets from V_3 to C_{33153} form a *C-set path*. 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_3, called a *C-set tree*, as shown in Figure 5(a).



(a) Tree template, K=2 (b) Tree realization, K=2 (c) Tree template, K=1

Figure 5: C-set tree examples

The above example is a special case of multiple joins, where the noti-sets of all nodes in W are the same (namely, V_3 in the example). Generally, the noti-sets of all nodes in W may not be the same. Then, nodes 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. Each C-set tree can be treated separately. Hence, in the balance of this subsection, our discussion is focused on a single C-set tree.

We next present formal definitions for a C-set tree. In what follows, we use l to denote one digit, $l \in [b]$, and $l_j...l_1$ to denote a string of j digits (we define $l_j...l_1$ to be the empty string if $j = 0$). Note that C-set trees are conceptual structures used for protocol design and reasoning about K-consistency. They are *not implemented* in any node.

Definition 4.7 *Suppose a set of nodes, $W = \{x_1, ..., x_m\}$, $m \geq 1$, join a K-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, W, and K, denoted by $C(V, W, K)$, 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 $W_{l_j...l_1\cdot\omega} \neq \emptyset$, $2 \leq j \leq d-k$, l_1,...,$l_j \in [b]$, then set $C_{l_j...l_1\cdot\omega}$ is a child of set $C_{l_{j-1}...l_1\cdot\omega}$.*

[9]That is, nodes in V_3, 14233 and 53013, need to update their neighbor tables when nodes in W join: each of them should update its $(1, 3)$-entry to store two neighbors with suffix 33 eventually; and each should update its $(1, 5)$-entry to store one neighbor with suffix 53 eventually.

Given V, W and K, the tree template is determined. The value of K affects the tree template through the noti-sets of nodes in W. Suppose $K = 1$ in the above example. Then, by Definition 4.4, nodes 41633 and 30633 have $\{14233\}$ as their noti-set, and node 33153 has $\{53013,\ 14233\}$ as its noti-set. And there would be two separate C-set trees instead of one, as shown in Figure 5(c).

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 and the C-set tree is realized. For instance, in the above example ($K = 2$), when 14233 updates its (1,3)-entry and fills 30633 into the entry, conceptually 30633 is filled into C_{33}. 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. We use $cset(V, W, K)$ to denote the C-set tree realized at the end of all joins, defined below, where $t^e = \max(t^e_{x_1}, ..., t^e_{x_m})$, as defined in Section 4.1.

Definition 4.8 *Suppose a set of nodes, $W = \{x_1, ..., x_m\}$, $m \geq 2$, join a K-consistent network $\langle V, \mathcal{N}(V)\rangle$, and for any node x, $x \in W$, $V_x^{Notify} = V_\omega$, $|\omega| = k$. Then the C-set tree realized at time t^e, denoted as $cset(V, W, K)$, is defined as follows:*

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

Intuitively, to obtain the C-set tree realized at the end of all joins, we take a snapshot of all of the neighbor tables at time t^e and construct a C-set tree realization as follows. First, for each node u, $u \in V_\omega$, and for each l_1 such that $l_1 \in [b]$ and $W_{l_1\cdot\omega} \neq \emptyset$, put all (k, l_1)-neighbors of u into $C_{l_1\cdot\omega}$, if u has such neighbors. Next, for each node v, $v \in C_{l_1\cdot\omega}$, and for each l_2 such that $l_2 \in [b]$ and $W_{l_2 l_1\cdot\omega} \neq \emptyset$, put all $(k+1, l_2)$-neighbors of v into $C_{l_2 l_1\cdot\omega}$, and so on. Note that in a C-set tree realization for $K = 1$, C-sets only contain nodes in W, while for $K \geq 2$, a C-set may also contain nodes in V_ω, the root set of the tree. Figure 5(b) shows one possible realization of the tree template in Figure 5(a). Observe that since for any node x, we set $N_x(i, x[i]).\mathit{first} = x$ for routing efficiency, $i \in [b]$, 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 14233 and 53013 store 30633 in $(1, 3)$-entry, then conceptually 30633 is filled in C_{33} and consequently, 30633 $\in C_{633}$, C_{0633} and C_{30633}.

The concept of C-set tree not only helps us in protocol design, but also guides us in reasoning about K-consistency. To prove that by the end of all joins, the neighbor tables have been constructed and updated such that they satisfy the K-consistency conditions, our approach is to prove the following **correctness conditions**, based on the C-set tree realization.

(1) $cset(V, W, K)$ has the same structure as $C(V, W, K)$. Also, for any C-set in $cset(V, W, K)$, say $C_{\omega'}$, it contains at least K nodes with suffix ω' if there exist at least K nodes in $(V \cup W)_{\omega'}$; otherwise, it contains all nodes in $(V \cup W)_{\omega'}$.

(2) For each node y, $y \in V_\omega$ (root of the C-set tree), and for each C-set $C_{l \cdot \omega'}$, $l \in [b]$, such that ω' is a suffix of $y.ID$, y has stored $\min(K, |C'_{l \cdot \omega}|)$ nodes with suffix $l \cdot \omega'$ in $N_y(k', l)$, where $k' = |\omega'|$.

(3) For each node x, $x \in W$, the C-set whose suffix is $x.ID$ is a leaf C-set in the tree. Let path-x denote the path from this leaf C-set to the root of the tree. Then, for any C-set, $C_{l \cdot \omega'}$, such that $C_{l \cdot \omega'}$ is a C-set along path-x, or a sibling C-set of a C-set along path-x, x has stored $\min(K, |C_{l \cdot \omega'}|)$ nodes with suffix $l \cdot \omega'$ in $N_x(k', l)$, $k' = |\omega'|$.

By the end of joins, if condition (1) is satisfied, then for every C-set that exists in the tree template (recall that given V, W, and K, the tree template is dertermined), it also exists in the tree realization and is not empty. Moreover, for each C-set in the tree realization, if there exist at least K nodes in $V \cup W$ that have the suffix of the C-set, then the C-set is filled with at least K nodes with the suffix; otherwise, all nodes in $V \cup W$ that have the suffix are included in the C-set. If conditions (1) and (2) are satisfied, then every table entry in the neighbor tables of nodes in V that needs to be updated has been updated and satisfies K-consistency conditions. If conditions (1) and (3) are satisfied, plus that each joining node has copied neighbor pointers from nodes in V, then for any joining node, its table has been constructed such that every table entry satisfies K-consistency conditions. Hence, the above three correctness conditions, together with each joining node's copying neighbors from nodes in V, ensure that the network is K-consistent after the joins.

5 Join Protocol for K-consistency

In this section, we present our design of a new join protocol that constructs and maintains K-consistent neighbor tables for an arbitrary number of nodes to join a network that initially is K-consistent. 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*.

A node with status *in_system* is called an **S-node**; otherwise, it is a **T-node**. Briefly, in status *copying*, a joining node, x, copies neighbor information from some S-nodes to construct most part of its table. In status *waiting*, x tries to "attach" itself to the network, i.e., to find an S-node that will store it as a neighbor, which indicates that conceptually it is filled into a C-set in the C-set tree. In status *notifying*, x seeks and notifies nodes that are conceptually in the subtree rooted at the parent set of the C-set x is filled into. Lastly, when it finds no more node to notify, x changes status to *in_system* and becomes an S-node.

5.1 Lowest attach-level

We first present an important concept, called *lowest attach-level*. We will discuss the cases where the concept is applied later in protocol specification.

In an 1-consistent network, a neighbor, say x, is only stored at one level in the table of a node y, given $x \neq y$. More specifically, x is only stored at level-k in $y.table$, where $k = |csuf(x.ID, y.ID)|$, since y itself is stored in $N_y(i, x[i])$ for all level-i, $0 \le i < k$ (both x and y have the required suffix for these entries). For $K \ge 2$, however, it is possible for y to store x at any level that is no higher than level-k. Thus, level-k is the highest level that x can be stored in $y.table$. In constructing a correctness proof for the join protocol, we found that a constraint on the lowest level that x can be stored in $y.table$ is also needed. We call it the *lowest attach-level* of x, or simply the *attach-level* of x for notational convenience.

Definition 5.1 *The* **attach-level** *of node x in the table of node y ($x \neq y$) is j, $0 \le j \le d-1$, determined as follows. (Let k denote $|csuf(x.ID, y.ID)|$.)*

- *$j = 0$ if $N_y(i, x[i]).size < K$ for all i, $0 \le i \le k$;*
- *$j = i$ if there exists a level i, such that $0 < i \le k$, $N_y(i-1, x[i-1]).size = K$, and $N_y(i', x[i']).size < K$ for all i', $i \le i' \le k$;*
- *an attach-level does not exist if $N_y(k, x[k]).size = K$.*

5.2 Protocol specification

Suppose a set of nodes W join a K-consistent network $\langle V, \mathcal{N}(V) \rangle$. Figure 6 presents the state variables of a joining node (a node in W). Note for each neighbor in its table, a node also stores the neighbor's state, which can be S indicating that the neighbor is in status *in_system* or T indicating that it is not yet. Variables in the first part in Figure 6 are also used by nodes in V, where initially for each node u, $u \in V$, $u.status$ = *in_system*, $u.table$ is populated with nodes in V in such a way that satisfies conditions in Definition 3.3, and $u.state(v) = S$ for every neighbor v that is stored in $u.table$. Figure 7 presents the protocol messages. Figures 8 to 12 present the pseudocode of the protocol, in which x, y, u and v denote nodes, and i, j and k denote integers. Note that when any node, x, stores a neighbor, say y, into $N_x(i, j)$, x needs to send a *RvNghNotiMsg*($y, x.state(y)$) to y if $y \neq x$, and y should reply to x if $x.state(y)$ is not consistent with $y.status$. For clarity of presentation, we have omitted the sending and reception of these messages in the pseudocode.

State variables of a joining node x:

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

$x.att\_level$: 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 6: State variables

Action in status *copying* In status *copying*, a joining node, x, fills most of its table entries by copying neighbor information from S-nodes, as follows. To construct its table at level-i, $i \in [d]$, x needs to find a node, g_i, that is an S-node and shares the rightmost i digits with it so that x can send a *CpRstMsg* to g_i

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 and request the receiver to store x, when $x.status$ is *waiting*.
JoinWaitRlyMsg(r, i, $x.table$), sent by x in response to a *JoinWaitMsg*, when $x.status$ is *in_system*. $r \in \{$*negative, positive*$\}$, i: an integer.
JoinNotiMsg(i, $x.table$), sent by x to notify receiver of the existence of x, when $x.status$ is *notifying*. i: an integer.
JoinNotiRlyMsg(r, Q, $x.table$, f), sent by x in response to a *JoinNotiMsg*. $r \in \{$*negative, positive*$\}$, Q: a set of integers, $f \in \{$*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 7: Protocol messages

to request a copy of $g_i.table$. We assume that each joining node knows a node in V. Let this node be g_0 for x. From $g_0.table$, x searches for a node that shares the rightmost digit with it and is an S-node. Let this node be g_1. x then contacts g_1 to request a copy of $g_1.table$. From $g_1.table$, x searches for a node, g_2, that shares the rightmost two digits with it and is an S-node, and so on. Figure 8 depicts the action in this status. The subroutine *Set_Neighbor()* is specified in Figure 13. (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.)

In status *copying*, each time after receiving a *CpRlyMsg*, x checks whether it should change status to *waiting*. Suppose x receives a *CpRlyMsg* from y. Then the condition for x to change status to *waiting* is: (i) There exists an attach-level for x in the copy of $y.table$ included in the reply, or (ii) an attach-level does not exist for x in the copy of $y.table$ but node u is a T-node, where $u = N_y(k, x[k]).\mathit{first}$ and $k = |\mathit{csuf}(x.ID, y.ID)|$. If the condition is satisfied, then x changes status to *waiting* and sends a *JoinWaitMsg* to y (case (i) holds) or to u (case (ii) holds). Otherwise, x remains in status *copying* and sends a *CpRstMsg* to u.

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

$i$: initially 0. $p$, $g$: a node, initially $g_0$. $s \in \{T, S\}$, initially $S$.

$x.status$ = copying;
for($i$ = 0; $i < d$; $i$++) {$N_x(i, x[i]).first = x$; $x.state(x) = T$;}
while ($g \neq$ null and $s == S$) { // copy level-$i$ neighbors of $g$
  $h = -1$; $k = |csuf(x.ID, g.ID)|$;
  while ($i \leq k \wedge h == -1$){
    for ($j$ = 0; $j < b$; $j$++)
      for (each $v$, $v \in N_g(i, j)$)
        for ($l = i$, $l \leq k$, $l$++) { Set_Neighbor($l$, $v[l]$, $v$, $g.state(v)$); }
    if ((for each $l$, $i \leq l \leq k$, $N_g(l, x[l]).size < K$) $\wedge h == -1$)
      { $p = g$; $g$ = null; $h = i$; }
    $i$++;
  }
  if ($h == -1$){ $p = g$; $g = N_p(k, x[k]).first$; $s = p.state(g)$;}
}
$x.status$ = waiting;
if ($g$ == null){Send JoinWaitMsg to $p$;$Q_n = Q_n \cup \{p\}$;$Q_r = Q_r \cup \{p\}$;}
else{ Send JoinWaitMsg to $g$; $Q_n = Q_n \cup \{g\}$; $Q_r = Q_r \cup \{g\}$; }
```

Figure 8: Action in status *copying*

Action in status *waiting* In status *waiting*, the main task of x is to find an S-node in the network to store x as a neighbor by sending out *JoinWaitMsg*; another task is to copy more neighbors into its table. The *JoinWaitMsg* x sends to a node, say y, serves as a notification to y that x is waiting to be stored in y's table. When y receives the *JoinWaitMsg* from x, there are two cases. (1) If y is still a T-node, it stores the message to be processed after it has become an S-node. (2) If y is an S-node, it checks whether there exists an attach-level for x in its table. If an attach-level exists, say level-j, y stores x into level-j through level-k, where $k = |\mathit{csuf}(x.ID, y.ID)|$ and $k \geq j$, and sends a *JoinWaitRlyMsg*(*positive*, j, $y.table$) to inform x that the lowest level x is stored is level-j. Level-j then becomes the **attach-level of x in the network**, stored by x in $x.att\_level$. If an attach-level does not exist for x, y sends *JoinWaitRlyMsg*(*negative*, -1, $y.table$) to x. After receiving the reply (positive or negative), x searches the copy of $y.table$ included in the reply for new neighbors to update its own table.

Upon receiving a negative reply from y, x has to send another *JoinWaitMsg*, this time to u, $u = N_y(k, x[k]).\mathit{first}$, $k = |\mathit{csuf}(x.ID, y.ID)|$.[10] This process may be repeated for several times (at most d times since each time the receiver shares at least one more digit with x than the previous receiver) until x receives a positive reply, which indicates that x has been stored by an S-node and therefore attached to the network. x then changes status to *notifying*. Figure 9 presents actions for a node upon receiving *JoinWaitMsg* and *JoinWaitRlyMsg*. Subroutines *Check_Ngh_Table()* and *Switch_To_S_Node()* are specified in Figure 13.

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

$k = |csuf(x.ID, y.ID)|$; $h = -1$; $j$ = 0;
if ($y.status$ == in_system) {
  while ($j \leq k \wedge h == -1$) {
    if (for each $l$, $j \leq l \leq k$, $N_y(l, x[l]).size < K$) {
      $h = j$; for ($l = j$; $l \leq k$; $l$++) { Set_Neighbor($l$, $x[l]$, $x$, $T$); }
    }else $j$++;
  }
  if ($h == -1$) Send JoinWaitRlyMsg(negative, $h$, $y.table$) to $x$;
  else Send JoinWaitRlyMsg(positive, $h$, $y.table$) to $x$;
}else $Q_j = Q_j \cup \{x\}$;

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

$Q_r = Q_r - \{y\}$; $k = |csuf(x.ID, y.ID)|$; $x.state(y) = S$;
if ($r$ == positive) {
  $x.status$ = notifying; $x.att\_level = i$;
  for ($j = i$; $j \leq k$; $j$++) { $R_x(j, x[j]) = R_x(j, x[j]) \cup \{y\}$; }
}else { // a negative reply, needs to send another JoinWaitMsg
  $v = N_y(k, x[k]).first$;
  Send JoinWaitMsg to $v$; $Q_n = Q_n \cup \{v\}$; $Q_r = Q_r \cup \{v\}$;
}
Check_Ngh_Table($y.table$);
if ($x.status$ == notifying $\wedge Q_r == \phi \wedge Q_{sr} == \phi$) Switch_To_S_Node();
```

Figure 9: Action on receiving JoinWaitMsg and JoinWaitRlyMsg

Action in status *notifying* In status *notifying*, x searches and notifies nodes that share the rightmost j digits with it, $j = x.att\_level$, so that these nodes will update their neighbor tables if necessary. x starts this process by sending *JoinNotiMsg*, which includes both $x.att\_level$ and a copy of $x.table$, to its neighbors at levels j and higher. Each *JoinNotiMsg* serves as a notification as well as a request for a copy of the receiver's table. Upon receiving a *JoinNotiMsg*, a receiver, z, stores x into all $(i, x[i])$-entries that are not full with K neighbors yet, where $x.att\_level \leq i \leq |\mathit{csuf}(x.ID, z.ID)|$, searches the copy of

[10] u can be any node in $N_y(k, x[k])$. We choose it to be $N_y(k, x[k]).\mathit{first}$ consistently in our protocol implementation.

$x.table$ for new neighbors to update z's table, and then replies to x with $z.table$ included in the reply. From the reply, if x find any node, say v, in $z.table$ such that v shares the right most j digits with x, $j = x.att\_level$, and if x has not sent *JoinNotiMsg* to v before, x will notify v by sending a *JoinNotiMsg* to it. Meanwhile, x searches the copy of $z.table$ for new nodes to update its own table. Figure 10 presents actions for a node on receiving *JoinNotiMsg* and *JoinNotiRlyMsg*.

So far, three cases for a node x to know another node y have been presented: (i) x copies y in status *copying*, (ii) x receives a *JoinWaitMsg* or a *JoinNotiMsg* 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 10 and 11. Suppose in status *notifying*, x sends a *JoinNotiMsg* to y. When y receives the message, if y is an S-node and finds that y is not included in $N_x(k, y[k])$, where $k = |csuf(x.ID, y.ID)|$, then y sets a flag f to be true in its reply. (Note that y is a qualified node for $N_x(k, y[k])$.) Seeing the flag in the reply, x sends a *SpeNotiMsg(x, y)* to u_1 to inform it about y if x has not done so and if $k > x.att\_level$, where $u_1 = N_x(k, y[k]).first$. If when u_1 receives the *SpeNotiMsg(x, y)* from x, its $(k_1, y[k_1])$-entry is already filled with K neighbors and y is not one of them, $k_1 = |csuf(u_1.ID, y.ID)|$, it forwards the message to u_2, where $u_2 = N_x(k_1, y[k_1]).first$. This process stops when a receiver stores or has stored y in its table and sends a *SpeNotiRlyMsg(x, y)* to x. (The process can be repeated at most d times.) Figure 11 depicts the actions on receiving *SpeNotiMsg* and *SpeNotiRlyMsg*.

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

 $Q$: a set of integers, initially empty

 $k = |csuf(x.ID, y.ID)|$; $f$ = false;
 for ($j = i$; $j \le k$, $j$++){ Set_Neighbor($j, x[j], x, T$);}
 for ($j = i$; $j \le k$, $j$++) {if ($x \in N_y(j, x[j])$) {$Q = Q \cup \{j\}$;}}
 if ($y \notin N_x(k, y[k]) \wedge y.status$ == in_system) $f$ = true;
 if ($Q \neq \emptyset$) Send JoinNotiRlyMsg(positive, $Q$, $y.table$, $f$) to $x$;
 else Send JoinNotiRlyMsg(negative, $\emptyset$, $y.table$, $f$) to $x$;
 Check_Ngh_Table($x.table$);

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

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

Figure 10: Action on receiving JoinNotiMsg and JoinNotiRlyMsg

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

 $k = |csuf(y.ID, u.ID)|$; Set_Neighbor($k, y[k], y, S$);
 if ($y \notin N_u(k, y[k])$)
   Send SpeNotiMsg($x, y$) to $N_u(k, y[k]).first$;
 else
   Send SpeNotiRlyMsg($x, y$) to $x$;


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

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

Figure 11: Action on receiving SpeNotiMsg and SpeNotiRlyMsg

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* and becomes an S-node. It then informs all of its reverse-neighbors, i.e., nodes that have stored x as a neighbor, that it has become an S-node. If x has delayed processing *JoinWaitMsg* from some nodes, it should process these messages and reply to these nodes at this time. Figure 12 and Figure 13 presents the peudocode for this part.

```
Action of $y$ on receiving a InSysNotiMsg from $x$:

 $y.state(x) = S$;
```

Figure 12: Action on receiving InSysNotiMsg

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

 for (each $u$, $u \in N_y(i, j) \wedge u \neq x, i \in [d], j \in [b]$) {
   $k = |csuf(x.ID, u.ID)|$; $s = y.state(u)$;
   for ($h = i$; $h \le k$; $h$++) { Set_Neighbor($h, u[h], u, s$); }
   if ($x.status$ == notifying $\wedge k \ge x.att\_level \wedge u \notin Q_n$) {
    Send JoinNotiMsg($x.att\_level$, $x.table$) to $u$;
    $Q_n = Q_n \cup \{u\}$; $Q_r = Q_r \cup \{u\}$;
   }
 }

Set_Neighbor($i, j, u, s$) at $x$:

 if ($u \neq x \wedge N_x(i, j).size < K \wedge u \notin N_x(i, j)$)
   { $N_x(i, j) = N_x(i, j) \cup \{u\}$; $x.state(u) = s$;}

Switch_To_S_Node() at $x$:

 $x.status$ = in_system; $x.state(x) = S$;
 for (each $v$ of $x$'s reverse neighbors) Send InSysNotiMsg to $v$;
 for (each node $u$, $u \in Q_j$) {
   $k = |csuf(x.ID, u.ID)|$; $h = -1$; $j$ = 0;
   while ($j \le k \wedge h == -1$){
    if (for each $l$, $j \le l \le k$, $N_x(l, u[l]).size < K$){
     $h = j$; for ($l = h$; $l \le k$; $l$++) { Set_Neighbor($l, u[l], u, T$); }
    }else $j$++;
   }
   if ($h \neq -1$) Send JoinWaitRlyMsg(positive, $h$, $x.table$) to $u$;
   else Send JoinWaitRlyMsg(negative, $h$, $x.table$) to $u$;
 }
```

Figure 13: Subroutines

6 Protocol Analysis

In this section, we present our correctness proof for the join protocol, and evaluate the protocol performance through both theoretical analysis and simulation experiments. We only present important lemmas and proof outlines in this section. Proof details are included in Appendix B.

6.1 Correctness of join protocol

We present two theorems. Suppose an arbitrary number of nodes join an initially K-consistent network by using the join protocol. Theorem 2 states that the join process of each node eventually terminates, and Theorem 3 states that at the end of joins, the resulting network is K-consistent. Recall that t^b_x denotes the starting time of the join duration of node x, t^e_x denotes the end of the join duration of x, and t^e denotes $\max(t^e_{x_1}, ..., t^e_{x_m})$.

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

Proof of Theorem 2: First, consider a joining node, x, in status *copying*. x eventually changes status to *waiting* because it sends at most d *CpRstMsg* and each receiver of a *CpRstMsg* replies to x with no waiting. Second, consider a joining node, x, in status *waiting*. In this status, x sends *JoinWaitMsg* to at most d nodes. We next show that for each *JoinWaitMsg* it sends out, 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 status *notifying* and will wait until it becomes an S-node before replying to x. Thus, to complete the proof, it suffices to show that any joining node in status *notifying* eventually becomes an S-node. Last, consider a joining node, z, in status *notifying*. There are two types of messages sent by z in this status, *JoinNotiMsg* and *SpeNotiMsg*. z only sends *JoinNotiMsg* to a subset of nodes in $V \cup W$ that share the rightmost i digits with itself, $i = z.att\_level$, 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.[11] Each *SpeNotiMsg* is forwarded at most d times before a reply is sent to z, and each receiver of the message replies to z or forwards the message to another node with no waiting. Therefore, z eventually becomes an S-node. ■

Theorem 3 *Suppose a set of nodes,* $W = \{x_1,...,x_m\}$, $m \geq 1$, *join a* K*-consistent network* $\langle V, \mathcal{N}(V)\rangle$. *Then, at time* t^e, $\langle V \cup W, \mathcal{N}(V \cup W)\rangle$ *is a* K*-consistent network.*

To prove Theorem 3, we first divide nodes in W into different groups, where nodes in the same group join *concurrently* and any two nodes that are in different groups join *sequentially*. Next, for each group of concurrent joins, we divide nodes in that group into several sub-groups, such that joins of nodes in the same sub-group are *dependent* while joins of any two nodes that are in different sub-groups are *independent*. (We will discuss how to do the node divisions later in this section.) We start by presenting Lemmas 6.1 to 6.4, which state the correctness of the join protocol for a single join, sequential joins, concurrent and independent joins, and concurrent and dependent joins.

Lemma 6.1 *Suppose node* x *joins a* K*-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 a* K*-consistent network.*

Lemma 6.2 *Suppose a set of nodes,* $W = \{x_1,...,x_m\}$, $m \geq 2$, *join a* K*-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 a* K*-consistent network.*

Lemma 6.3 *Suppose a set of nodes,* $W = \{x_1,...,x_m\}$, $m \geq 2$, *join a* K*-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* K*-consistent.*

Lemma 6.4 *Suppose a set of nodes,* $W = \{x_1,...,x_m\}$, $m \geq 2$, *join a* K*-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* K*-consistent.*

To prove Lemma 6.4, consider any two nodes in W, say x and y. If their noti-sets are the same, i.e., $V_x^{Notify} = V_y^{Notify}$, then x and y belong to the same C-set tree rooted at V_x^{Notify}, otherwise they belong to different C-set trees. We consider nodes in the same C-set tree first. To simplify presentation in the following propositions, we make the following assumption:

[11]In simulations, we observed that *SpeNotiMsg* is rarely sent.

Assumption 1 *(for Propositions 6.1 to 6.7) A set of nodes,* $W = \{x_1, ..., x_m\}$, $m \geq 2$, *join a* K*-consistent network* $\langle V, \mathcal{N}(V)\rangle$ *concurrently and for any* x, $x \in W$, $V_x^{Notify} = V_\omega$ *and* $|\omega| = k$.

Propositions 6.1 states that every joining node is filled into some C-set in the C-set tree by the end of joins. Note that ω is the suffix of the root set in the C-set tree, as stated in Assumption 1. Propositions 6.2 and 6.3 state that correctness conditions (1) and (2), stated in Section 4, are satisfied by time t^e, respectively. (Recall that $l_j...l_1$ denotes the empty string if $j = 0$.) Proposition 6.4 states that for a C-set that node x belongs to ($l = l_j$), or a sibling C-set of a C-set x belongs to ($l \neq l_j$), x eventually stores enough neighbors with the suffix of that C-set. For instance, consider the example in Figure 5(b) and let $x = 41633$. By Proposition 6.4, for any C-set of C_{633}, C_{1633}, C_{41633}, and C_{0633} (the former three are the C-sets x belongs to, and C_{0633} is a sibling C-set of C_{1633}), say C_{633}, eventually x stores $\min(K, H)$ neighbors in its $(2, 6)$-entry, where H is the total number of nodes with suffix 633 in $V \cup W$. Proofs of the propositions are based on induction upon the C-set tree realized at time t^e.

Proposition 6.1 *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* $x.ID$.

Proposition 6.2 *If* $W_{l_j...l_1\cdot\omega} \neq \emptyset$, $1 \leq j \leq d-k$, *then by time* t^e, *the followings are true:*

(a) $C_{l_j...l_1\cdot\omega} \subseteq (V \cup W)_{l_j...l_1\cdot\omega}$ *and* $C_{l_j...l_1\cdot\omega} \supseteq V_{l_j...l_1\cdot\omega}$.
(b) if $|(V \cup W)_{l_j...l_1\cdot\omega}| < K$, *then* $C_{l_j...l_1\cdot\omega} = (V \cup W)_{l_j...l_1\cdot\omega}$;
(c) if $|(V \cup W)_{l_j...l_1\cdot\omega}| \geq K$, *then* $|C_{l_j...l_1\cdot\omega}| \geq K$.

Proposition 6.3 *Consider any node* x, $x \in V_\omega$. *For any C-set* $C_{l\cdot l_j...l_1\cdot\omega}$, $0 \leq j \leq d-k-1$ *and* $l \in [b]$, *if* $l_j...l_1 \cdot \omega$ *is a suffix of* $x.ID$, *then* $N_x(k+j, l).size = \min(K, |(V \cup W)_{l\cdot l_j...l_1\cdot\omega}|)$ *holds by time* t^e.

Proposition 6.4 *For any C-set,* $C_{l_j...l_1\cdot\omega}$, $1 \leq j \leq d-k$, l_1,...,$l_j \in [b]$, *the following assertion holds by time* t^e: *For each* x, $x \in C_{l_j...l_1\cdot\omega}$ *and* $x \in W$, $N_x(k+j-1, l).size = \min(K, |(V \cup W)_{l\cdot l_{j-1}...l_1\cdot\omega}|)$, $l \in [b]$.

For any node x, $x \in W$, we define **the first C-set** x **belongs to** for x in a C-set tree realization to be (i) $C_{l_1\cdot\omega}$ if $x \in C_{l_1\cdot\omega}$; (ii) $C_{l_j...l_1\cdot\omega}$ for $j > 1$, if $x \in C_{l_j...l_1\cdot\omega}$ and $x \notin C_{l_{j-1}...l_1\cdot\omega}$.

Proposition 6.5 states that for any ancestor C-set of the first C-set node x belongs to (or for any sibling C-set of such an ancestor C-set), x eventually stores enough neighbors with the suffix of that C-set (or of that sibling C-set). For instance, consider again the example in Figure 5(b) and node 41633. The first C-set 41633 belongs to is C_{633}. There is one ancestor C-set of C_{633}, C_{33}, which also has a sibling C-set, C_{53}. Then by Proposition 6.5, 41633 has stored $\min(K, |(V \cup W)_{33}|)$ neighbors in its $(1, 3)$-entry by time t^e; moreover, 41633 has stored $\min(K, |(V \cup W)_{53}|)$ neighbors in its $(1, 5)$-entry by time t^e.

Based on Propositions 6.4 and 6.5, we prove Proposition 6.6, which states that correctness condition (3), stated in Section 4, is satisfied by time t^e. Note that in Propositions 6.3 and 6.6, $l \cdot l_j...l_1 \cdot \omega$ is the required suffix of the $(k+j, l)$-entry in $x.table$, where $k = |\omega|$. Next, based on Propositions 6.2, 6.3, and 6.6, we prove Proposition 6.7, which states that by time t^e, every table entry in the network satisfies K-consistency conditions and hence the network is K-consistent. (Recall that Propositions 6.1 to 6.7 are stated under the assumption, Assumption 1.)

Proposition 6.5 *For any x, $x \in W$, suppose $C_{l_j ... l_1 \cdot \omega}$ is the first C-set x belongs to, where $l_j ... l_1 \cdot \omega$ is a suffix of $x.ID$, $1 \leq j \leq d-k$. Then for any i, $0 \leq i \leq j$, and any l, $l \in [b]$, $N_x(k+i,l).size = \min(K, |(V \cup W)_{l \cdot l_i ... l_1 \cdot \omega}|)$.*

Proposition 6.6 *For any node x, $x \in W$, if $(V \cup W)_{l \cdot l_j ... l_1 \cdot \omega} \neq \emptyset$, where $l_j ... l_1 \cdot \omega$ is a suffix of $x.ID$, $0 \leq j \leq d-k-1$, and $l \in [b]$, then $N_x(k+j,l).size = \min(K, |(V \cup W)_{l \cdot l_j ... l_1 \cdot \omega}|)$ holds by time t^e.*

Proposition 6.7 *For each node x, $x \in V \cup W$, $N_x(i+k,j).size = \min(K, |(V \cup W)_{j \cdot x[i-1] ... x[0]}|)$ holds by time t^e, $i \in [d]$, $j \in [b]$.*

So far, we have proved correctness of the join protocol for the case where a set of nodes join dependently and all joining nodes belong to the same C-set tree. Next, Proposition 6.8 extends the result to joining nodes that belong to different C-set trees. It states that for any joining node, say x, for any suffix that exists in a different C-set tree other than the one x belongs to, if the suffix is also the required suffix of a table entry in $x.table$, then eventually x has stored enough neighbors in that table entry. (Note that in Proposition 6.8, $l \cdot \omega_2$ is the required suffix for the (k_2, l)-entry in $x.table$.) Based on the propositions, we can then prove Lemma 6.4 and Lemma 6.5.

Proposition 6.8 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, join a K-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 $\omega_1 \neq \omega_2$ and ω_2 is a suffix of ω_1. Let $k_2 = |\omega_2|$. Then, by time t^e, for any x, $x \in G(V_{\omega_1})$, the following assertion holds: $N_x(k_2, l).size = \min(K, |(V \cup W)_{l \cdot \omega_2}|)$, $l \in [b]$.*

Proof of Lemma 6.4: . (Outline) 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, $x \in G(V_{\omega_i})$ if and only if $V_x^{Notify} = V_{\omega_i}$, $1 \leq i \leq h$. Then, by Propositions 6.3, 6.7, and 6.8, the lemma holds. ■

Lemma 6.5 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, join a K-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 a K-consistent network.*

Proof of Lemma 6.5: (Outline) 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 $G_1 = \emptyset$):

1. Pick any node x, $x \in W - \bigcup_{j=1}^{i-1} G_j$, and put x in G_i.
2. For each node y, $y \in W - \bigcup_{j=1}^{i} G_j$,
   (a) if there exists a node z, $z \in G_i$, such that $(V_y^{Notify} \cap V_z^{Notify} \neq \emptyset)$, then put y in G_i; or
   (b) if there exists a node z, $z \in G_i$, and a node u, $u \in G_i$, such that the following is true: $(V_y^{Notify} \subset V_u^{Notify}) \wedge (V_z^{Notify} \subset V_u^{Notify})$, then put y in G_i; or
   (c) if there exists a node z, $z \in G_i$, and a node u, $u \in W - \bigcup_{j=1}^{i} G_j$, such that the following is true: $(V_y^{Notify} \subset V_u^{Notify}) \wedge (V_z^{Notify} \subset V_u^{Notify})$, then put both y and u in G_i.
3. Increment i and repeat steps 1 to 3 until $\bigcup_{j=1}^{i} G_j = W$.

Then, we get groups $\{G_i, 1 \leq i \leq l\}$. It can be checked that $V_x^{Notify} \cap V_y^{Notify} = \emptyset$ for any node x, $x \in G_i$, and any node y, $y \in G_j$, where $1 \leq i \leq l$, $1 \leq j \leq l$, and $i \neq j$. By Lemmas 6.3 and 6.4, the lemma holds. ■

Proof of Theorem 3: If $m = 1$, then by Lemma 6.1, the theorem holds.

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 $[t_{G_i}^b, t_{G_i}^e]$, $1 \leq i \leq l$, 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, if $|G_1| \geq 2$, by Lemma 6.5, at time $t_{G_1}^e$, $\langle V \cup G_1, \mathcal{N}(V \cup G_1) \rangle$ is a K-consistent network; if $|G_1| = 1$, then by Lemma 6.1, $\langle V \cup G_1, \mathcal{N}(V \cup G_1) \rangle$ is a K-consistent network at time $t_{G_1}^e$. Similarly, by applying Lemma 6.5 (or Lemma 6.1) to G_2, ..., G_l, we conclude that eventually, at time t^e, $\langle V \cup W, \mathcal{N}(V \cup W) \rangle$ is a K-consistent network. ■

6.2 Protocol performance

We first analyze the communication cost of each join. Here we only present results for the number of messages of type *CpRstMsg*, *JoinWaitMsg*, and *JoinNotiMsg*,[12] since these messages may include a copy of a neighbor table and thus could be big in size. The other types of messages are all small in size. (See Figure 7.) Ananlysis of numbers of small messages can be found in Appendix C. In general, the number of each type of the small messages is at most $O(\log n)$, and some of these messages can be piggy-backed by probing messages to reduce the cost.

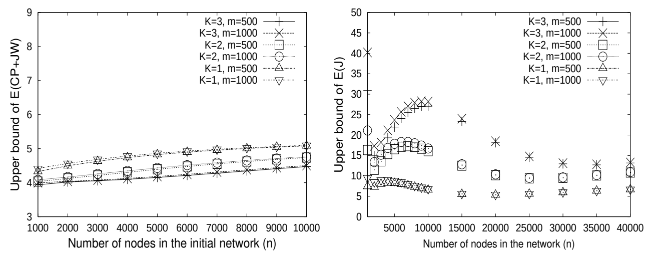
Let $C(X, Y)$ denote the number of Y-combinations of X objects, n denote the number of nodes in the initial network, and m denote the number of joining nodes. Moreover, we define two functions, $Q_i(r)$ and $P_i(r)$, to be used in Theorems 4 to 6, where $Q_i(r) \geq K$, $0 \leq P_i(r) \leq 1$, and $\sum_{i=0}^{d-1} P_i(r) = 1$. We note that when $b^d \gg r$, $Q_i(r)$ is approximately $K + \frac{r}{b^i}$

Definition 6.1 *Let $P_i(r)$ denote a function defined as follows, where r and i denote integers, $r \geq 1$ and $0 \leq i \leq d-1$.*

- *If $1 \leq r < K$, then $P_i(r) = 1$ for $i = 0$ and $P_i(r) = 0$ for $1 \leq i \leq d-1$;*
- *If $r \geq K$, then*
  - $P_i(r) = \frac{\sum_{j=0}^{K} C(b^{d-1}-1,j) C(b^d - b^{d-1}, r-j)}{C(b^d-1,r)}$ *for $i = 0$;*
  - $P_i(r) = \frac{\sum_{j=0}^{K} C(b^{d-1-i}-1,j) \sum_{k=K-j}^{\min(r-j,B)} C(B,k) C(b^d - b^{d-i}, r-k-j)}{C(b^d-1,r)}$ *where $B = (b-1)b^{d-i-1}$, for $1 \leq i < d-1$;*
  - $P_i(r) = 1 - \sum_{j=0}^{d-2} P_j(r)$ *for $i = d-1$.*

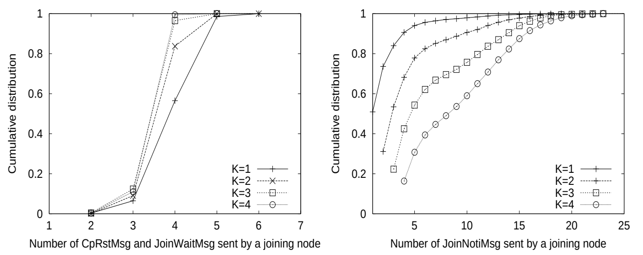
Definition 6.2 *Let $Q_i(r)$ denote a function defined as follows, where r and i denote integers, $r \geq 1$ and $0 \leq i \leq d-1$.*

[12] The number of replies to these messages are the same since requests and replies are one-to-one related.



(a) CpRstMsg + JoinWaitMsg (b) JoinNotiMsg

Figure 14: Theoretical upper bound of expected number of messages vs. n, for different values of K and m, $b = 16$, $d = 40$



(a) CpRstMsg+JoinWaitMsg (b) JoinNotiMsg

Figure 15: Cumulative distribution of messages sent by a joining node, $n = 3200$, $m = 800$, $b = 16$, $d = 40$

- *If* $1 \le r < K$, *then* $Q_i(r) = r$;
- *If* $r \ge K$, *then*
  $Q_i(r) = K + \frac{\sum_{j=0}^{\min(r,D)} C(D,j)C(b^d - b^{d-i}, r-j)}{C(b^d - K - 1, r)}$
  where $D = b^{d-i} - K - 1$.

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

Theorem 5 *Suppose node* x *joins a* K*-consistent network* $\langle V, \mathcal{N}(V)\rangle$, $|V| = n$. *Then, the expected number of JoinNotiMsg sent by* x *is* $\sum_{i=0}^{d-1} Q_i(n-K)P_i(n) - 1$.

Theorem 6 *Suppose a set of nodes,* $W = \{x_1,...,x_m\}$, $m \ge 1$, *join a* K*-consistent network* $\langle V, \mathcal{N}(V)\rangle$, $|V| = n$. *Then for any node* x, $x \in W$, *an upper bound of the expected number of JoinNotiMsg sent by* x *is* $\sum_{i=0}^{d-1} Q_i(n+m-1-K)P_i(n)$.

Proofs of the above theorems are presented in Appendix C. Here we only present the intuitions for proving Theorem 5. Suppose $V_x^{Notify} = V_\omega$. Since only node x joins, x needs to send *JoinNotiMsg* to all nodes in V_x^{Notify}, except the one it sends *JoinWaitMsg* to. Let X denote the number of nodes in V_ω, i.e., $X = |V_\omega|$. Then the number of *JoinNotiMsg* x sends out is $X - 1$. Let $Y = |\omega|$ and $P(Y = i)$ denote the probability of $Y = i$. To compute $E(X-1)$, we have $E(X) = E(E(X|Y)) = \sum_{i=0}^{d-1}(E(X|Y = i)P(Y = i))$. It can then be proved that $E(X|Y = i) = Q_i(n-K)$ and $P(Y = i) = P_i(n)$, where $n = |V|$.

Figure 14 plots the upper bounds presented in Theorem 4 and Theorem 6, where $E(CP + JW)$ is the expected number of *CpRstMsg* and *JoinWaitMsg* sent by a joining node, and $E(JN)$ is the expected number of *JoinNotiMsg*. Notice that for a fixed value of K, both upper bounds are insensitive to the value of m (number of joins), and increase very slightly as n becomes large. Moreover, for the same values of n and m, the upper bound of $E(JN)$ increases when K value increases, while the upper bound of $E(CP + JW)$ decreases when K value increases.

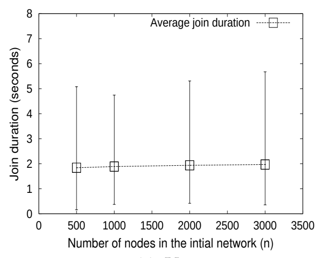
Next, we study performance of the protocol through simulation experiments. We have implemented our join protocol in detail in an event-driven simulator. To generate network topologies, we used the GT_ITM package [15]. We simulated the sending of a message and the reception of a message as events, but abstracted away queueing delays. The end-to-end delay of a message from its source to destination was modeled as a random variable with mean value proportional to the shortest path length in the underlying network. For the experiments reported in this section, a topology of 2112 routers was used, with 4000 nodes (end hosts) randomly attached to the routers. The end-to-end delays were in the range of 0 to 329 ms, with the average being 113 ms. In each simulation, we let all joins start at the same time, which maximizes the number of nodes that join concurrently and dependently and thus maximizes the average join durations.

Figure 15 summarizes results from experiments where 800 nodes joined a network that initially had 3,200 nodes. Figure 15(a) shows simulation results of cumulative distribution of the number of *CpRstMsg* and *JoinWaitMsg* sent by joining nodes, and Figure 15(b) shows results of cumulative distribution of the number of *JoinNotiMsg* sent by joining nodes. As shown in the figure, the number of *CpRstMsg* and *JoinWaitMsg* sent by a joining node is small, which is less than seven in Figure 15(a). Moreover, majority of joining nodes sent a small number of *JoinNotiMsg*. For example, in Figure 15(b), for $K = 3$, more than 75% joining nodes sent less than ten *JoinNotiMsg*.[13]

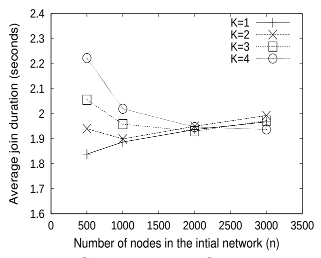
Both the theoretical analysis and simulation results show that when the value of K increases, communication cost also increases. (Besides the number of *JoinNotiMsg*, numbers of small messages also increase with K [4].) Clearly, there is a tradeoff between benefits and maintenance overhead of a K-consistent network for different K values. Detailed study of the tradeoff is presented in [5].

Lastly, we study lengths of join durations through simulation experiments. For each simulation setup, we ran five experiments to obtain the average join durations. Figure 16(a) presents average join durations for 1000 nodes joining networks of different sizes (different values of n), where $K = 1$. Each errorbar shows the minimum and maximum join durations observed in the five experiments for that simulation setup. Figure 16(b) presents the average join duration as a function of n, for different values of K. From the results, we observe that the average join duration is short in general, and increases very slightly when n increases (in some cases, e.g., for $K = 4$, it even decreases when n increases).

[13]For the results shown in Figure 15(a), the average number of *CpRstMsg* and *JoinWaitMsg* sent by a joining node was 4.381 for $K = 1$, 4.071 for $K = 2$, 3.907 for $K = 3$, and 3.892 for $K = 4$; the corresponding theoretical upper bounds are 4.68, 4.25, 4.07, and 4.017, respectively. For the results shown in Figure 15(b), the average number of *JoinNotiMsg* was 6.714 for $K = 1$, 11.649 for $K = 2$, 13.971 for $K = 3$, and 14.751 for $K = 4$; the corresponding theoretical upper bounds are 8.636, 14.924, 18.033, and 19.842, respectively.



(a) $K = 1$



(b) Average join durations

Figure 16: Join durations, $m = 1000$, $b = 16$, $d = 40$

7 Network Initialization

To initialize a K-consistent network of n nodes, we can put any one of the nodes, say x, in V, and construct $x.table$ as follows.

- $N_x(i, x[i]).first = x$, $x.state(x) = S$, $i \in [d]$.
- $N_x(i, j) = \emptyset$, $i \in [d]$, $j \in [b]$ and $j \neq x[i]$.

Next, let the other $n - 1$ nodes join the network by executing the join protocol, each is given x to start with. Then, when all of the joins terminate, a K-consistent network is constructed.

8 Related Work

In PRR [11], a static set of nodes and pre-existence of consistent and optimal neighbor tables are assumed. CAN [12] and Pastry [13] each has join, leave, and failure recovery protocols, but the issue of neighbor table consistency was not explicitly addressed. In Chord [14], maintaining consistency of neighbor tables ("finger tables" in Chord) was considered difficult in the presence of concurrent joins in a large network. A stabilization protocol was designed to maintain consistency of just one neighbor pointer per node ("successor pointer"), which is sufficient to guarantee correctness of object location.

In Tapestry [3], a join protocol was presented with a proof of correctness for concurrent joins. Their 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 on a list (one list per table entry being updated) until it has received acknowledgments from all downstream nodes. In their approach, many existing nodes have 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.

Storing several qualified nodes in each neighbor table entry was first suggested in PRR [11] to facilitate the location of replicated objects. In Tapestry [16], storing two backup neighbors in addition to the primary neighbor in each table entry (that is, $K = 3$) was recommended for fault-tolerance and to improve hypercube routing performance. However, these papers do not have the K-consistency concept. Therefore they provide neither protocols to construct K-consistent neighbor tables nor any theoretical analysis of the benefits of K-consistency.

9 Conclusions

For the hypercube routing scheme used in several proposed p2p systems [11, 13, 16, 7], we introduced the property of K-consistency, and showed that K-consistent neighbor tables are resilient even when a large fraction of nodes in the network fail. We then presented the detailed specification of a new join protocol for the scheme. Furthermore, we presented a conceptual foundation, C-set trees, for guiding our protocol design and reasoning about K-consistency. By induction on C-set trees, we proved that the new join protocol generates K-consistent neighbor tables for an arbitrary number of concurrent joins. The expected communication cost of integrating a new node into the network is shown to be small by both theoretical analysis and simulations. The join protocol presented in this paper can also be used to initialize a K-consistent network.

An observation from a companion study [5] is that networks in which each node maintains a larger number of consistent neighbor pointers are not only more resilient, but they also *recover more quickly and completely* from node failures. From our analytic and simulation results, we found that the improvement in network resilience from $K = 1$ to $K = 2$ is dramatic. We conclude that hypercube routing networks should be K-consistent with $K \geq 2$. However, the larger the K, the higher is the maintenance overhead. Thus, we recommend a K value of 2 or 3 for p2p networks with a high rate of node dynamics; for p2p networks with a low rate of node dynamics, K may be higher than 3 if additional route redundancy is desired.

References

[1] 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*, June 2002.

[2] R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of DHT routing geometry on resilience and proximity. In *Proc. of ACM SIGCOMM*, August 2003.

[3] 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*, August 2002.

[4] S. S. Lam and H. Liu. Silk: a resilient routing fabric for peer-to-peer networks. Technical Report TR-03-13, Dept. of CS, Univ. of Texas at Austin, May 2003.

[5] S. S. Lam and H. Liu. Failure recovery for structured p2p networks: Protocol design and performance evaluation. In *Proc. of ACM SIGMETRICS*, June 2004.

[6] S. S. Lam and H. Liu. Failure recovery for structured p2p networks: Protocol design and performance under churn. *Computer Networks*, Vol.50(No.16), November 2006.

[7] 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*, October 2002.

[8] H. Liu and S. S. Lam. Neighbor table construction and update in a dynamic peer-to-peer network. Technical Report

TR-02-46, Dept. of CS, Univ. of Texas at Austin, September 2002.

[9] H. Liu and S. S. Lam. Neighbor table construction and update in a dynamic peer-to-peer network. In *Proc. of IEEE International Conference on Distributed Computing Systems (ICDCS)*, May 2003.

[10] H. Liu and S. S. Lam. Consistency-preserving neighbor table optimization for p2p networks. In *Proc. of International Conference on Parallel and Distributed Systems*, July 2004.

[11] 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*, June 1997.

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

[13] 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*, November 2001.

[14] 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*, August 2001.

[15] E. W. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proc. of IEEE Infocom*, March 1996.

[16] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, Vol.22(No.1), January 2004.

Appendix

A Proofs of Lemmas 3.2, 3.3

Proof of Lemma 3.2: We prove the lemma by constructing K disjoint paths from x to y. Consider $N_x(0, y[0])$. $y \notin x.table$ implies $y \notin N_x(0, y[0])$. Hence, there must exist K neighbors in $N_x(0, y[0])$; otherwise, $N_x(0, y[0]).size < K$ implies $|V_{y[0]}| < K$ and all nodes in $V_{y[0]}$, including y, would be stored in $N_x(0, y[0])$.

We denote the K paths to be constructed as P_0 to P_{K-1}. Also, we use u_i^j to denote the jth node in path P_i. According to Definition 3.2, we need to establish paths as follows: $P_i = \{u_i^0, ..., u_i^k\}$, $i \in [K]$, $1 \le k \le d$, where $u_i^0 = x$, $u_i^k = y$, and $u_i^j \in N_{u_i^{j-1}}(j-1, y[j-1])$, $1 \le j \le k$. First, let $u_i^0 = x$ for each path P_i, $i \in [K]$. Next, starting with P_0, for each path P_i, let $u_i^1 = v$, such that $v \in N_x(0, y[0])$ and $v \notin P_l$ for all l, $0 \le l \le i-1$, that is, v is not included in paths P_0 to P_{i-1} (this is easy to achieve since there are K nodes in $N_x(0, y[0])$). Let $j = 1$, $f = \min(K, |V_{y[j]...y[0]}|)$, and execute the following steps (referred to as round j).

1. For each path P_i, $i \in [K]$, if $u_i^j = y$, then mark P_i as "done". Let $P' = \{P_i,\ P_i$ is not marked "done"$\}$ and $|P'| = I$. Note $I \le K$. In the next three steps, we will assign a node to u_i^{j+1} for each path P_i in P'.
2. For each P_i, $P_i \in P'$, if $u_i^j[j] = y[j]$ then let $u_i^{j+1} = u_i^j$. Suppose there are h such paths. Then, re-number these paths as P_0 to P_{h-1}, and the other paths in P' as P_h to P_{I-1}. Then, for any path P_i, $h \le i \le I-1$, we have $u_i^j[j] \neq y[j]$. In the next two steps, we will assign a node to u_i^{j+1} for each path P_i in $\{P_h, P_{h+1}, ..., P_{I-1}\}$.
3. If $f \ge I$, then starting with P_h, for each path P_i, $h \le i \le I-1$, let $u_i^{j+1} = v$, such that $v \in N_{u_i^j}(j, y[j])$ and $v \neq u_l^{j+1}$ for all l, $0 \le l \le i-1$. Such a node v must exist, since there are f different nodes in $N_{u_i^j}(j, y[j])$, and at most $I-1$ of them are already assigned to other paths in P' (where there are $I-1$ paths other than P_i) for the $(j+1)$th position.
4. If $f < I$, then (i) starting with P_h, for path P_i, $h \le i \le f-1$, let $u_i^{j+1} = v$, such that $v \in N_{u_i^j}(j, y[j])$ and $v \neq u_l^{j+1}$ for all l, $0 \le l \le i-1$, and (ii) for each path P_i, $f \le i \le I-1$, let $u_i^{j+1} = y$, because $f < I$ indicates $f < K$, i.e., $|V_{y[j]...y[0]}| < K$, so every node in $V_{y[j]...y[0]}$, including y, is in $N_{u_i^j}(j, y[j])$.

Next, increase j by 1 and execute the above four steps for another round if there still exist paths that are not marked "done" yet. Eventually, each path will be marked "done", since the network is a K-consistent network, and a path exists from any node (including node u_i^1, $i \in [K]$) to y (see Lemma 3.1).

So far we have established K paths from x to y. We then prove that they are disjoint. First, we point out that any two paths, say P_i and P_j, among the K paths are different from each other, since at least u_i^1 is different from u_j^1.

Second, we need to prove the following claim, which states that for any two paths, the nodes at the jth position are different if none of the nodes is the destination node y.

Claim A.1 *For any two paths P_i and P_l, if $u_i^j \neq y$ and $u_l^j \neq y$, $j \ge 1$, then $u_i^j \neq u_l^j$.*

Proof of Claim A.1: Prove by induction. Base step ($j = 1$): According to the way we assign nodes to $u_{i'}^1$ for each path $P_{i'}$, $i' \in [K]$, we know that $u_i^1 \neq u_l^1$.

Inductive step: Suppose $u_i^j \neq u_l^j$, $j \ge 1$, where $u_i^j \neq y$ and $u_l^j \neq y$. We next prove that $u_i^{j+1} \neq u_l^{j+1}$ if neither u_i^{j+1} nor u_l^{j+1} is y.

- If $u_i^j[j] = y[j]$ and $u_l^j[j] = y[j]$, then according to step 2 in each round of path construction, $u_i^{j+1} = u_i^j$ and $u_l^{j+1} = u_l^j$, thus $u_i^{j+1} \neq u_l^{j+1}$.
- If $u_i^j[j] \neq y[j]$ or $u_l^j[j] \neq y[j]$, then without loss of generality, suppose $u_l^j[j] \neq y[j]$. Also, suppose in this round of node assignment (round $j+1$), path P_i is re-numbered as $P_{i'}$ (see step 2), path P_l is re-numbered as $P_{l'}$, and $i' < l'$ (if $u_i^j[j] = y[j]$, then according to step 2, we have $i' < l'$; otherwise, we suppose $i' < l'$). Let $v = u_i^{j+1}$. According to step 3 (or 4) in path construction, if $u_l^{j+1} \neq y$, then u_l^{j+1} is chosen in such a way that it is not the same as any $(j+1)$th node in the 0th path to the l'th path (the paths that are re-numbered as the 0th path to the l'th path in round

$j+1$). Hence, $u_l^{j+1} \neq v$, i.e., $u_l^{j+1} \neq u_i^{j+1}$. ■

Third, by Claim A.1, we can show (by contradiction) that among the K paths we have constructed, no path is of the form $(x, ..., z, ..., x, ..., y)$, where $z \neq x$. Suppose there exists a path P_i of the above form, that is, there exists a path P_i such that for the nodes in P_i, $u_i^0 = x$, $u_i^j = z$, and $u_i^{j+1} = x$, where $j > 0$. $u_i^{j+1} = x$ indicates that $x.ID$ shares the rightmost $j+1$ digits with $y.ID$, then, $x[0] = y[0]$ and $x \in N_x(0, y[0])$. Hence, there must exist a path P_l such that $u_l^1 = x$ (according to the way we assign nodes to $u_{i'}^1$ for each path $P_{i'}$). Thus, P_l is not the same path with P_i. Then, by step 2, in path P_l, $u_l^1 = ... = u_l^j = u_l^{j+1} = x$. Next, by Claim A.1, for any other path P_h, $h \neq l$, $u_h^{j'} \neq u_l^{j'}$ for $1 \leq j' \leq j+1$. Hence, no j'th node in any path other than P_l could be node x for $1 \leq j' \leq j+1$. We conclude with $u_i^{j+1} \neq x$, which contradicts with the assumption $u_i^{j+1} = x$.

Based on the above results, we prove that the K paths are disjoint. Consider any two paths P_i and P_l. By Claim A.1, $u_i^j \neq u_l^j$, that is, the jth node in P_i is different from the jth node in P_l. We next show that u_i^j is different from any j'th node in P_l, $j' < j$, by contradiction. Suppose $u_i^j = u_l^{j'}$. Then since u_i^j has suffix $y[j]...y[0]$, so does $u_l^{j'}$. According to step 2 in path construction, $u_l^{j'} = u_l^{j'+1} = ... = u_l^j$. Thus, we get $u_i^j = u_l^j$, a contradiction. Similarly, we can prove that u_i^j is different from any j'th node in P_l, for $j' > j$. Therefore, any node in P_i that is not x or y does not appear in any other path P_l. Thus, the K paths are disjoint. ■

Proof of Lemma 3.3: By Lemma 3.2, if $y \notin x.table$, then there exist at least K disjoint paths from x to y. Also, as shown in the proof of Lemma 3.2, if $y \notin x.table$, then $N_x(0, y[0]).size = K$ and thus $\min(K, |V_{y[0]}|) = K$. Hence, the lemma holds when $y \notin x.table$. If $y \in x.table$, however, $y \notin N_x(0, x[0])$, then, $N_x(0, y[0]).size = \min(K, |V_{y[0]}|)$. Similar to the proof for Lemma 3.2, we can construct h disjoint paths from x to y, where $h = \min(K, |V_{y[0]}|)$. If $y \in x.table$ and $y \in N_x(0, x[0])$, then $y[0] = x[0]$. Recall that $x \in N_x(0, x[0])$. Similar to the proof for Lemma 3.2, we can construct $h-1$ paths from x to y, $h = \min(K, |V_{y[0]}|)$, where in assigning nodes to u_i^1 for each path, we only consider the nodes in set N', $N' = N_x(0, x[0]) - \{x\}$. (If we also consider x in assigning nodes to u_i^1, two of the paths maybe the same path that goes directly from x to y: path P_i, where $u_i^1 = x$ and path P_l where $u_l^1 = y$.) Hence, at least $h-1$ disjoint paths exist from x to y. ■

B Proofs of Lemmas 6.1 to 6.5

In this section, we present our proof for the lemmas presented in Section 6.1 in detail. Recall that we made the following assumptions in designing the join protocol: (i) The initial network is a K-consistent network, (ii) each joining node, by some means, knows a node in the initial network initially, (iii) messages between nodes are delivered reliably, and (iv) there is no node deletion (leave or failure) during the joins. We also assume that the actions specified in Figures 8, 9, 10, 11, and 12 are atomic.

Theorem 3 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 1$, join a K-consistent network $\langle V, \mathcal{N}(V)\rangle$. Then, at time t^e, $\langle V \cup W, \mathcal{N}(V \cup W)\rangle$ is a K-consistent network.*

To prove Theorem 3, we first prove some auxilary lemmas and propositions. Table 2 shows the abbreviations we will use for protocol messages in the proofs, and Table 3 presents the notation used in the following proofs. Moreover, we define "strongly reachable" as follows.

Definition B.1 *Consider two nodes, x and y, in network $\langle V, \mathcal{N}(V)\rangle$. If there exists a neighbor sequence (a path), $(u_h, u_{h+1}..., u_k)$, $0 \leq h \leq k \leq d$, such that $u_h = x$, $u_k = y$, and $u_{i+1} \in N_{u_i}(i, y[i])$, $h \leq i \leq k-1$, where $h = |csuf(x.ID, y.ID)|$, then we say that y is* **strongly reachable** *from x, or x can* **strongly reach** *y, in k hops.*

Protocol Message	***Abbreviation***
CpRlyMsg	*CPRly*
JoinWaitMsg	*JW*
JoinWaitRlyMsg	*JWRly*
JoinNotiMsg	*JN*
JoinNotiRlyMsg	*JNRly*
SpeNotiMsg	*SN*
SpeNotiRlyMsg	*SNRly*
RvNghNotiMsg	*RN*
RvNghNotiRlyMsg	*RNRly*

Table 2: Abbreviations for protocol messages

Notation	***Definition***
$\langle x \longrightarrow y\rangle_k$	x can strongly reach y within k hops
$x \overset{j}{\rightarrow} y$	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
$A(x)$	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 3: Notation in proofs

The following facts, which can be easily observed from the join protocol, are used frequently in the proofs. (In what follows, unless explicitly stated, when we say "x can reach y", we mean "x can strongly reach y".)

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

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

Fact B.3 *If $A(x) = u$, then $x.att\_level \leq h$, where $h = |csuf(x.ID, u.ID)|$, and for each j, $x.att\_level \leq j \leq h$, $x \in N_u(h, x[h])$ after u receives the* JW *from x. Also, x changes status from waiting to notifying immediately after it receives the positive* JWRly *from u.*

Fact B.4 *If $A(x) = u$ and $x.att\_level = k$, $0 \leq k \leq |csuf(x.ID, u.ID)|$, then before u receives a* JW *from x, $N_u(j, x[j]).size < K$ for all j, $k \leq j \leq |csuf(x.ID, u.ID)|$.*

Fact B.5 *A joining node, x, only sends a* JN *to y if x is in status notifying and $|csuf(x.ID, y.ID)| \geq x.att\_level$.*

Fact B.6 *If $x \overset{jn}{\rightarrow} y$ happens, y will send a reply that includes $y.table$ to x immediately. Moreover, each* JN *sent by x includes $x.table$.*

Fact B.7 *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 B.8 *By time t^e_x, x has received all of the replies for messages of type* CP, JW, JN, *and* SN *it has sent out.*

Proposition B.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 $|csuf(x.ID, z.ID)| \geq x.att\_level$, then $x \xrightarrow{j} z$ happens before time t^e_x.

Proof: Since at time t, y can reach z, there must exist a neighbor sequence at time t, (u_h, u_{h+1}, ..., u_d), $h = |csuf(y.ID, z.ID)|$, such that $u_h = y$, $u_d = z$, and $u_{i+1} \in N_{u_i}(i, z[i])$ 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: *For nodes in* {u_h, u_{h+1}, ..., u_d}*, If $x \xrightarrow{jn} y$ happens, then $x \xrightarrow{jn} u_i$ eventually happens for each i, $h+1 \leq i \leq d$.*

First, observe that at time t, x is still in status *waiting*, if $x \xrightarrow{jn} y$ happens, it must happen after time t, by Facts B.4. Let $k = x.att\_level$. If $x \xrightarrow{jn} y$ happens (i.e., x sends a *JN* to y, then it must be that $k \leq |csuf(x.ID, u_i.ID)|$, by Fact B.5. Therefore, y must share the suffix $x[k-1]...x[0]$ with x. On the other hand, it is given that $|csuf(x.ID, z.ID)| \geq k$, thus z also shares suffix $x[k-1]...x[0]$ with x. Since both y and z have suffix $x[k-1]...x[0]$ in their IDs, it follows that each node along the path from y to z, {u_h, u_{h+1}, ..., u_d} shares suffix $x[k-1]...x[0]$. Thus, $k \leq |csuf(x.ID, u_i.ID)|$ for each i, $h \leq i \leq d$. Then, if $x \xrightarrow{jn} u_i$ happens, $h \leq i \leq d-1$ from the *JNRly* u_i sends to x, x finds u_{i+1} from the reply and then sends a *JN* to u_{i+1}. Thus, the above claim is true.

Therefore, if $x \xrightarrow{jn} y$ happens, then eventually $x \xrightarrow{jn} u_d$ will happen, where $u_d = z$. The proposition holds in the first case.

If $y = u$, that is, y is the attching-node of x, then by Fact B.3, $k \leq |csuf(x.ID, y.ID)|$. From the *JWRly* y sends to x, x will find u_{h+1} and sends a *JN* to u_{h+1}. Then similar to the above argument, it can be shown that $x \xrightarrow{jn} u_i$ eventually happens, $h+1 \leq i \leq d$. Therefore, $x \xrightarrow{jn} u_d$ will happen, $u_d = z$. ■

Lemma 6.1 *Suppose node x joins a K-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 a K-consistent network.*

Proof: Suppose $V^{Notify}_x = V_{x[k-1]...x[0]}$, that is, $|V_{x[k]...x[0]}| < K$ and $|V_{x[k-1]...x[0]}| \geq K$. Let $V' = V \cup \{x\}$. Then $V'_{j \cdot x[i-1]...x[0]} = V_{j \cdot x[i-1]...x[0]}$ if $j \neq x[i]$, $i \in [d]$, and $V'_{x[i]...x[0]} = V_{x[i]...x[0]} \cup \{x\}$.

Let g be the last node that x sends a *CP* to in status *copying*. Then it must be that $g \in V_{x[k-1]...x[0]}$: Because the condition for x to change status is that x finds there exists a level-h in the table of g, such that $N_g(i, x[i]).size < K$, for all $h \leq i \leq |csuf(x.ID, g.ID)|$. And since $V_{x[k-1]...x[0]} \geq K$, $V_{x[k]...x[0]} < K$, and $\langle V, \mathcal{N}(V)\rangle$ is K-consistent, then before x is stored in any other node's table, $N_g(i, x[i]).size \geq K$ for $0 \leq i \leq k-1$, and $N_g(k, x[k]).size < K$. Therefore, by copying neighbor information from nodes in V, by the time x changes status to *waiting*, $N_x(i, j).size = \min(K, |V_{j \cdot x[i-1]...x[0]}|) = \min(K, |V'_{j \cdot x[i-1]...x[0]}|)$ if $j \neq x[i]$; if $j = x[i]$ and $0 \leq i < k$, then $N_x(i, j).size = K$ since $|V_{j \cdot x[i-1]...x[0]}| \geq K$; for $(i, x[i])$-entry, $k \leq i \leq d-1$, for any node y, if $y \in V_{x[i]...x[0]}$, then $y \in N_x(i, x[i])$. Moreover, since $x \in N_x(i, x[i])$, $i \in [d]$, it follows that for $k \leq i \leq d-1$, $N_x(i, x[i]) = V_{x[i]...x[0]} \cup \{x\} = V'_{x[i]...x[0]}$. Therefore, entries in $x.tabe$ satisfy the conditions in Definition 3.3.

After x changes status from *copying* to *waiting*, it sends a *JW* to node g, which will then store x in $N_x(k, x[k])$ (and levels higher than k if x and g share a suffix that is longer than $x[k-1]...x[0]$) and sends back a positive *JWRly*. Thus, $x.att\_level = k$. Next, x needs to notify any node z, $z \in V_{x[k-1]...x[0]}$ about its join. Since the initial network is K-consistent, thus $\langle g \rightarrow z\rangle_d$ at the time g sends the positive *JWRly* to x. By Proposition B.1, $x \xrightarrow{j} z$ eventually happens. Therefore, eventually, $N_z(i, x[i]) = V_{x[i]...x[0]} \cup \{x\}$, i.e., $N_z(i, x[i]) = V'_{x[i]...x[0]}$, $k \leq i \leq |csuf(x.ID, z.ID)|$. The other entries remain unchanged. It is trivial to check that the unchanged entries satisfy conditions in Definition 3.3 for the new network. ■

Corollary B.1 *Suppose a set of nodes, W ={x_1,...,x_m}, $m \geq 1$, join a K-consistent network $\langle V, \mathcal{N}(V)\rangle$. Then for any node x, $x \in W$, by time t^e_x, $N_x(i, j).size = K$ if $|V_{j \cdot x[i-1]...x[0]}| \geq K$; and $N_x(i, j) \supseteq V_{j \cdot x[i-1]...x[0]}$ if $|V_{j \cdot x[i-1]...x[0]}| < K$.*

Corollary B.2 *Suppose a set of nodes, W ={x_1,...,x_m}, $m \geq 1$, join a K-consistent network $\langle V, \mathcal{N}(V)\rangle$. Then for any node x, $x \in W$, and any node y, $y \in V$, $\langle x \rightarrow y\rangle_d$ by time t^e_x.*

Lemma 6.2 *Suppose a set of nodes, W ={x_1,...,x_m}, $m \geq 2$, join a K-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 a K-consistent network.*

Proof: Prove by induction on $t^e_{x_i}$, $1 \leq i \leq m$. By Lemma 6.1, Lemma 6.2 holds when $i = 1$. Assume when $1 \leq i < m$, Lemma 6.2 holds. Then at time $t^e_{x_i}$, $\langle V \cup W', \mathcal{N}(V \cup W')\rangle$ is a K-consistent network, where $W' = \{x_1, ..., x_i\}$. Since the nodes join sequentially, $t^b_{x_{i+1}} \geq t^e_{x_i}$. Thus, when x_{i+1} joins, the network, which is composed of nodes in $V \cup W'$, is K-consistent and there is no other joins in the period of $[t^b_{x_{i+1}}, t^e_{x_{i+1}}]$. By Lemma 6.1, at time $t^e_{x_{i+1}}$, $\langle V \cup \{x_1, ..., x_{i+1}\}, \mathcal{N}(V \cup \{x_1, ..., x_{i+1}\})\rangle$ is K-consistent. Hence, Lemma 6.2 also holds for $i+1$. ■

Lemma B.1 *Suppose a set of nodes, $W = \{x_1, ..., x_m\}$, $m \geq 2$, join a K-consistent network $\langle V, \mathcal{N}(V)\rangle$ independently. For any node x, $x \in W$, if $|V_{j \cdot x[i-1]...x[0]}| < K$, $0 \leq i < d-1$, $j \in [b]$, then $(V \cup W')_{j \cdot x[i-1]...x[0]} = V_{j \cdot x[i-1]...x[0]}$, where $W' \subseteq W - \{x\}$.*

Proof: We prove by contradiction. Assume $(V \cup W')_{j \cdot x[i-1]...x[0]} \supset V_{j \cdot x[i-1]...x[0]}$. Then there exists a least a node y such that $y \in W'$ and $y.ID$ has suffix $j \cdot x[i-1]...x[0]$.

Since $|V_{j\cdot x[i-1]...x[0]}| < K$ and $j \cdot x[i-1]...x[0]$ is a suffix of $y.ID$, we rewrite it as $|V_{y[i]y[i-1]...x[0]}| < K$. Let $V_y^{Notify} = V_{y[i'-1]...y[0]}$. Then by the definition of V_y^{Notify}, we know $|V_{y[i']y[i'-1]...y[0]}| < K$. Therefore, we know $i' \leq i$. Since $y[i-1]...y[0] = x[i-1]...x[0]$ and $i' \leq i$, we know $y[i'-1]...y[0] = x[i'-1]...x[0]$.

Now consider V_x^{Notify}. Suppose $V_x^{Notify} = V_{x[j-1]...x[0]}$. If $1 \leq j \leq i'$, then $V_{x[j-1]...x[0]} \supset V_{x[i'-1]...x[0]}$; if $i' < j \leq d-1$, then $V_{x[j-1]...x[0]} \subset V_{x[i'-1]...x[0]}$. Thus, $V_{x[j-1]...x[0]} \cap V_{x[i'-1]...x[0]} \neq \emptyset$, i.e., $V_{x[j-1]...x[0]} \cap V_{y[i'-1]...y[0]} \neq \emptyset$. Then we get $V_x^{Notify} \cap V_y^{Notify} \neq \emptyset$. However, by Definition 4.5, $V_x^{Notify} \cap V_y^{Notify} = \emptyset$. Contradiction. ■

Corollary B.3 *Suppose a set of nodes, $W = \{x_1, ..., x_m\}$, join a K-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, $x \in G(V_{\omega_1})$, $(V \cup G(V_{\omega_2}))_{j\cdot x[i-1]...x[0]} = V_{j\cdot x[i-1]...x[0]}$ if $|V_{j\cdot x[i-1]...x[0]}| < K$.*

Lemma 6.3 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, join a K-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 K-consistent.*

Proof: Consider any node x, $x \in W$. If $|V_{j\cdot x[i-1]...x[0]}| \geq K$, then by Corollary B.1, by time t^e, $N_x(i,j).size = K$. If $|V_{j\cdot x[i-1]...x[0]}| < K$, then by Lemma B.1, we have $(V \cup W)_{j\cdot x[i-1]...x[0]} = V_{j\cdot x[i-1]...x[0]}$ for $j \neq x[i]$, and $(V \cup W)_{j\cdot x[i-1]...x[0]} = V_{j\cdot x[i-1]...x[0]} \cup \{x\}$ for $j = x[i]$, $i \in [d]$ and $j \in [b]$. Then, by Corollary B.1, $N_x(i,j).size = |V_{j\cdot x[i-1]...x[0]}|$ for $j \neq x[i]$; and $N_x(i,j).size = |V_{j\cdot x[i-1]...x[0]}| + 1$ for $j = x[i]$, where $N_x(i,j) = V_{j\cdot x[i-1]...x[0]} \cup \{x\}$. Therefore, entries in the table of x satisfy conditions in Definition 3.3.

Next, consider any node y, $y \in V$, and the (i,j)-entry in $y.table$, $i \in [d]$ and $j \in [b]$. If $|V_{j\cdot y[i-1]...y[0]}| \geq K$, then $N_y(i,j).size = K$ since the initial network is K-consistent. If $|V_{j\cdot y[i-1]...y[0]}| < K$ and $W_{j\cdot y[i-1]...y[0]} = \emptyset$, then $N_y(i,j) = V_{j\cdot y[i-1]...y[0]} = (V \cup W)_{j\cdot y[i-1]...y[0]}$. If $|V_{j\cdot y[i-1]...y[0]}| < K$ and $W_{j\cdot y[i-1]...y[0]} \neq \emptyset$, then there exists a node x, $x \in W$, such that $j \cdot y[i-1]...y[0]$ is a suffix of x. By Lemma B.1, x is the only node in W has the suffix $j \cdot y[i-1]...y[0]$. Similar to the argument in proving Lemma 6.1, we can prove that $x \xrightarrow{j} y$ happens before time t_x^e. Hence, $N_y(i,j) = V_{j\cdot y[i-1]...y[0]} \cup \{x\} = (V \cup W)_{j\cdot y[i-1]...y[0]}$.

The above results are true for every node in W. Hence, by time t^e, $\langle V \cup W, \mathcal{N}(V \cup W)\rangle$ is a K-consistent network. ■

Proposition B.2 *Suppose a set of nodes, $W = \{x_1, ..., x_m\}$, $m \geq 1$, join a K-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 $x \xrightarrow{j} y$ happens, then by 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. Also, let $h = |csuf(x.ID, y.ID)|$.

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

(2) If at time t_i, $N_{u_i}(h_i, x[h_i]).size < K$ and $x \notin N_{u_i}(h_i, x[h_i])$, $h_i = |csuf(x.ID, u_i.ID)|$, then u_i stores x into $N_{u_i}(h, x[h])$. Hence, $\langle y \rightarrow x\rangle_d$, since a neighbor sequence from y to x, (u_0, u_1, ..., u_i, x), exists, where $u_0 = y$.

(3) If at time t_i, $N_{u_i}(h_i, x[h_i]).size = K$ and $x \notin N_{u_i}(h_i, x[h_i])$, then from u_i's reply (either a *JWRly* or a *JNRly*, both includes $u_i.table$), 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+1}. If $x \xrightarrow{jn} u_i$ happens, then x is in status *notifying* at time t_{i+1} and since $h_{i+1} \geq h_i \geq x.att\_level$, x needs to send a *JN* to u_{i+1}; if $x \xrightarrow{jw} u_i$ happens, then x is in status *waiting* at time t_{i+1} and needs to send u_{i+1} a *JW*. Therefore, $x \xrightarrow{j} u_{i+1}$ eventually happens (before t_x^e).

(4) Increment i and repeat steps (1) to (4).

We claim that steps (1) and (4) are 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, eventually there exists a node, u_j, $1 \leq i < d-h$, such that $x \in N_{u_j}(h_i, x[h_j])$, where $h_j = |csuf(x.ID, u_j.ID)|$. Therefore, eventually, there exists a neighbor sequence from y to x, which is (u_0, u_1, ..., u_j, x), where $u_0 = y$. 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, $\langle y \rightarrow x\rangle_d$. ■

Before we present Proposition B.3, we introduce a concept, contact-chain(y, u). Suppose $y \xrightarrow{j} u$ or $y \xrightarrow{c} u$ happens. Then we can construct a chain of nodes that y contacts after it sends out the message to u. We begin with the case $y \xrightarrow{j} u$ happens.

If $y \xrightarrow{j} u$ happens, then **contact-chain(y, u)** is a sequence of nodes, constructed as follows: Let $u_0 = u$, $i = 0$, and put u_0 in the chain initially. Let $|csuf(u_i.ID, y.ID)| = h_i$, $i \geq 0$.

(1) If after u receives the message from y, $y \in N_{u_i}(h_i, y[h_i])$, then u_i is the last node in the chain.

(2) If after u receives the message from y, $y \in N_{u_i}(h_i, y[h_i])$, then let $u_{i+1} = N_{u_i}(h_i, y[h_i]).first$. Add u_{i+1} to the chain. (It can be shown that $y \xrightarrow{j} u_{i+1}$ eventually happens.) Increment i and repeat the two steps.

Similarly, if $y \xrightarrow{c} u$ happens, then there also exists a chain of nodes, (u_0, u_1, ..., u_j), $j \geq 0$, such that $u_0 = u$, and y requests neighbor tables from u_0 to u_j, where $u_{i+1} = N_{u_i}(i, y[i]).first$, and y finds that an attach-level for it exists in the copy of $u_j.table$ in the *CPRly* from u_j. y then sends a *JW* to u_j. Concatenate (u_0, u_1, ..., u_j) with contact-chain(y, u_j), we get contact-chain(y, u_0), i.e., contact-chain(y, u).

Proposition B.3 *If $y \xrightarrow{j} u$ or $y \xrightarrow{c} u$ happens, then there exists a contact-chain(y, u).*

Proof: If $y \xrightarrow{j} u$, and if after u receives the message from y, $y \in N_u(h, y[h])$, then $\{u\}$ is the contact-chain, where $h = |csuf(y.ID, u.ID)|$.

Otherwise, let $i = 0$ and $u_0 = u$, and suppose after u_i receives the message from y, $y \in N_{u_i}(h_i, y[h_i])$. Let $h_i = |csuf(y.ID, u_i.ID)|$ and $u_{i+1} = N_{u_i}(h_i, y[h_i]).first$.

First, we show that $y \xrightarrow{j} u_{i+1}$ eventually will happen. The reason is as follows. (1) If the message y sent to u_i is *JN*, then it must be that $h_i \geq y.att\_level$. u_{i+1} shares more digits with y than u_i does. Hence, $h_{i+1} \geq h_i \geq y.att\_level$. Therefore, after y knows u_{i+1} from u_i's reply, it will send a *JN* to u_{i+1}. If the message y sent to u_i is *JW*, then u_i must reply y with a negative *JWRly* since it didn't store y. According to the join protocol, y will send out another *JW*, this time to u_{i+1}.

Second, we show that there exists a last node in the chain. That is, the step that after $y \xrightarrow{j} u_i$, y is not stored by u_i, and y sends another message to u_{i+1} ($y \xrightarrow{j} u_i$) will not be repeated infinitely. 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.

Similarly, we can show that a contact-chain(y, u) exists if $y \xrightarrow{c} u$ happens. ■

Proposition B.4 *Suppose a set of nodes, $W = \{x_1, ..., x_m\}$, $m \geq 2$, join a K-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, $x \xrightarrow{j} u$ has happened, and $y \xrightarrow{j} u$ or $y \xrightarrow{c} u$ has happened. If $|csuf(x.ID, y.ID)| = h$ and $x.att\_level \leq h$, then by time t_{xy}, $t_{xy} = \max(t^e_x, t^e_y)$, at least one of the following is true: $x \in N_y(h, x[h])$ or $N_y(h, x[h]).size = K$.*

Proof:
Case 1: $|csuf(u.ID, x.ID)| \geq h$. Let the time u replies to x be t_x, and the time u replies to y be t_y.

If $t_x < t_y$, then after receiving the notification from x (i.e., time t_x), u will store x in $N_u(h, x[h])$ if $N_u(h, x[h]).size < K$ before t_x ($x.att\_level \leq h$, hence u can store x at level h). Since $t_x < t_y$, at time t_y, either $x \in N_u(h, x[h])$ or $N_u(h, x[h]).size = K$ is true. Next, from u's reply that includes $u.table$, y copies nodes in $N_u(h, x[h])$ (after time t_y but before time t_{xy}). Thus, either $x \in N_y(h, x[h])$ or $N_y(h, x[h]).size = K$ by time t_{xy}.

If $t_x > t_y$, then consider the nodes y contacts after it sends the *CP* message to u, i.e., *contact-chain(y,u)*. Suppose *contact-chain(y,u)* is ($u_0, u_1, ..., u_f, u_{f+1}$), where $u_0 = u$ and $u_{f+1} = y$. Then, for each node in the chain, u_i, either $y \xrightarrow{c} u_i$ or $y \xrightarrow{j} u_i$ happens, $0 \leq i \leq f$. Observe that $|csuf(x.ID, u_i.ID)| \geq h$ (because each $u_i.ID$ has suffix $x[h-1]...x[0]$ since both $u_0.ID$ and $y.ID$ have this suffix), therefore, $|csuf(x.ID, u_i.ID)| \geq x.att\_level$ for each i, $0 \leq i \leq f$. We then prove the following claim:

Claim B.1 *(Property of contact-chain(y, u)) If after y has received all replies from u_0 to u_i and copied nodes from neighbor tables included in the replies, $N_y(h, x[h]).size < K$ and $x \notin N_y(h, x[h])$, then $x \xrightarrow{j} u_{i+1}$ happens eventually, $0 \leq i \leq f$.*

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 $u_{i+1} \in u_i.table$ by time t.

Proof of Claim B.1: Base step At time t_y, link (u_0, u_1) already exists (otherwise, $u_1 = y$). Therefore, the link also exists at time t_x (we have assumed $t_x > t_y$). x then learns y from u_0's reply. If the reply is a *JNRly*, then $x \xrightarrow{jn} u_1$ eventually happens because $x.att\_level \leq h$ (by the assumption of the proposition); if the reply is a *JWRly*, then x will send another *JW* to u_1, that is $x \xrightarrow{jw} u_1$ will happen. Thus, $x \xrightarrow{j} u_1$ eventually happens.
Inductive step Assume the claim holds for all j, $0 \leq j \leq i$, $0 \leq i \leq m - 1$. 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 $t_1 < t_2$, otherwise, at time t_1, either $x \in N_{u_{i+1}}(h, x[h])$ or $N_{u_{i+1}}(h, x[h]).size = K$ is true, which implies after y copies nodes from u_{i+1}'s reply, either $x \in N_y(h, x[h])$ or $N_y(h, x[h]).size = K$ is true, which contradicts with the assumption of the claim. Hence, link (u_{i+1}, u_{i+2}) exists at time t_1 as well as t_2. Consequently, x knows u_{i+2} from u_{i+1}'s reply and will notify u_{i+1} if it has not done so (similar to the argument in the base step, x sends either a *JW* or a *JN* to u_{i+1}). ■

It can then be shown that if after receiving all of the replies from u_0 to u_f, $N_y(h, x[h]).size < K$ and $x \notin N_y(h, x[h])$, then eventually $x \xrightarrow{j} y$ happens. Thus, the proposition holds in Case 1.

Case 2: $|csuf(u.ID, x.ID)| < h$. Then, it follows that $|csuf(u.ID, x.ID)| = |csuf(u.ID, y.ID)|$. Let $|csuf(u.ID, x.ID)| = h'$, then $x[h'] = y[h']$, since $x[h-1]...x[0] = y[h-1]...y[0]$ and $h' < h$. Let the time u receives the message from x (either a *JW* or a *JN*) be t_1, and the time u receives the message from y (a *CP*, *JW*, or a *JN*) be t_2.

(1) If $t_1 < t_2$, and $x \in N_u(h', x[h'])$ after t_1, then from u's reply to y, y finds x and copies x into $y.table$ (if y sends a *CP* or a *JW* to u) or $y \xrightarrow{jn} x$ happens. Hence, after y receives the reply from u, $x \in N_y(h, x[h])$ or $N_y(h, x[h]).size = K$.

(2) If $t_1 < t_2$, and $x \notin N_u(h', x[h'])$ after t_1, then $N_u(h', x[h'])$ has stored K nodes by time t_1. Let $v = N_u(h', x[h']).first$. Then $x \xrightarrow{j} v$ will happen (if the message x sent to u is *JN*, then $x \xrightarrow{jn} v$ happens; otherwise, $x \xrightarrow{jw} v$ happens). Similarly, $y \xrightarrow{j} v$ or $y \xrightarrow{c} v$ will happen, since at time $t_2 > t_1$ and $N_u(h', x[h'])$ already stores K nodes by t_1.

(3) If $t_1 > t_2$, and $y \in N_u(h', x[h'])$ by time t_1, then x finds y from u's reply. Then $x \xrightarrow{jn} y$ will happen since $x.att\_level \leq h$ (either that (1) x copies y into $x.table$ and sends a *JN* to y later, if x has sent a *JW* to u, or (2) x sends a *JN* to y right after it receives the *JNRly* from u).

(4) If $t_1 > t_2$, $y \notin N_u(h', x[h'])$ by time t_1, and the message y sends to u is a *JW* or *JN*, then $N_u(h', x[h'])$ must have stored K nodes by time t_2 (otherwise, u would store y at time t_2). Let $v = N_u(h', x[h']).first$. Then both $x \xrightarrow{j} v$ and $y \xrightarrow{j} v$ eventually happen.

(5) If $t_1 > t_2$, $y \notin N_u(h', x[h'])$ by time t_1, the message y sends to u is a *CP*, and $N_u(h', x[h'])$ has stored K nodes by time t_2, then $x \xrightarrow{j} v$ and $y \xrightarrow{c} v$ eventually happen.

(6) If $t_1 > t_2$, $y \notin N_u(h', x[h'])$ by time t_1, the message y sends to u is a *CP*, and $N_u(h', x[h'])$ has not stored K nodes by time t_2, then y must send a *JW* to u after it receives the *CPRly*

from u. Then, it is the same with the case that $x \xrightarrow{j} u$ and $y \xrightarrow{j} u$ both happen.

In (2), (4), and (5), we have that both $x \xrightarrow{j} v$ and $y \xrightarrow{c} v$ happen. Moreover, v shares more digits with x and y than u. If $|csuf(v.ID, x.ID)| \geq h$, then by applying the arguments in Case 1 (replacing u with v), we can show that the proposition holds. If $|csuf(v.ID, x.ID)| < h$, then arguments in Case 2 can be applied, where either we conclude that the proposition holds in (1), (3) and (6), or we get that $x \xrightarrow{j} v'$ and $y \xrightarrow{c} v'$ happen, where v' shares more digits with x and y than v. In the latter case, we repeat the above steps repeatedly until at a step, we find a node w, such that $x \xrightarrow{j} w$ and $y \xrightarrow{c} w$ both happen and $|csuf(w.ID, x.ID)| \geq h$. Then by applying the arguments in Case 1 (by replacing u with w), we conclude that the proposition holds. ■

Lemma 6.4 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, join a K-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 K-consistent.*

To prove Lemma 6.4, consider any two nodes in W, say x and y. If their noti-sets are the same, i.e., $V_x^{Notify} = V_y^{Notify}$, then x and y belong to the same C-set tree rooted at V_x^{Notify}, otherwise they belong to different C-set trees. We consider nodes in the same C-set tree first and prove Propositions 6.1 to 6.7. Then, we prove Proposition 6.8, which states when joining nodes belong to different C-set trees, their neighbor tables eventually satisfy K-consistency conditions. Based on Proposition 6.7 and Proposition 6.8, we present our proof of Lemma 6.4. To simplify presentation in the following propositions, we make the following assumption:

Assumption 1 *(for Propositions 6.1 to 6.7)*
A set of nodes, $W = \{x_1, ..., x_m\}$, $m \geq 2$, join a K-consistent network $\langle V, \mathcal{N}(V)\rangle$ concurrently and for any x, $x \in W$, $V_x^{Notify} = V_\omega$ and $|\omega| = k$.

Proposition 6.1 *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 *contact-chain(x,g)*, where g is the node that x is given to start its join process. Suppose *contact-chain(x,g)* is $(u_0, u_1, ...u_f, u_{f+1})$, where $u_0 = g$ and $u_{f+1} = x$. Then u_f is the node that sends a positive *JWRly* to x (see Definition of a *contact-chain* [8]). Let the lowest level u_f stores x in $u_f.table$ (the attach-level of x) be level-h, then $k \leq h \leq |csuf(u.ID, x.ID)|$ (recall $k = |\omega|$, as defined in Assumption 1). Create a new sequence $(g_0, ..., g_h)$ based on *contact-chain(x,g)* as follows:

- Let $g_0 = g$ and $j = 0$.
- For each i, $0 \leq i \leq h-1$, let $g_{i+1} = g_i$ if $g_i[i] = x[i]$ and $i < h-1$; if $g_i[i] \neq x[i]$ and $i < h-1$, let $g_i = u_j$ and increase j.
- $g_h = u_f$.

Then, $g_k \in V_\omega$, because $g_k \in V$ and $g_k[k-1]...g_k[0] = x[k-1]...x[0]$. Hence, $g_{k+1} \in C_{l_1\cdot\omega}$, where $l_1 = x[k]$, since $g_{k+1} \in N_{g_k}(k, x[k])$ (by the definition of *contact-chain*) and $g_{k+1}[k] = x[k]$. Consequently, $g_{k+2} \in C_{l_2 l_1\cdot\omega}$, ..., $g_{h-1} \in C_{l_{h-k-1}...l_1\cdot\omega}$, and $g_h \in C_{l_{h-k}...l_1\cdot\omega}$. Hence $x \in C_{x[h]\cdot l_{h-k}...l_1\cdot\omega}$. ■

Corollary B.4 *For each node x, $x \in W$, there exists a node u such that $u = A(x)$, and u belongs to a C-set in $cset(V, W)$ or $u \in V_\omega$.*

Proposition 6.2 *If $W_{l_j...l_1\cdot\omega} \neq \emptyset$, $1 \leq j \leq d-k$, then by time t^e, the followings are true:*

(a) $C_{l_j...l_1\cdot\omega} \subseteq (V \cup W)_{l_j...l_1\cdot\omega}$ and $C_{l_j...l_1\cdot\omega} \supseteq V_{l_j...l_1\cdot\omega}$.
(b) if $|(V \cup W)_{l_j...l_1\cdot\omega}| < K$, then $C_{l_j...l_1\cdot\omega} = (V \cup W)_{l_j...l_1\cdot\omega}$;
(c) if $|(V \cup W)_{l_j...l_1\cdot\omega}| \geq K$, then $|C_{l_j...l_1\cdot\omega}| \geq K$.

Proof: Consider set $C_{l_j...l_1\cdot\omega}$. For any node u, $u \in V_\omega$, if $u.ID$ has suffix $l_j...l_1 \cdot \omega$, then $u \in C_{l_j...l_1\cdot\omega}$ by the definition of $cset(V, W)$. Hence, part (a) holds trivially.

We prove parts (b) and (c) by contradiction. Assume $|C_{l_j...l_1\cdot\omega}| < h$, where $h = |(V \cup W)_{l_j...l_1\cdot\omega}|$ if $|(V \cup W)_{l_j...l_1\cdot\omega}| < K$, and $h = K$ if $|(V \cup W)_{l_j...l_1\cdot\omega}| \geq K$. If $|C_{l_j...l_1\cdot\omega}| < h$, then there exists a node x, such that $x \in W_{l_j...l_1\cdot\omega}$ and $x \notin C_{l_j...l_1\cdot\omega}$. By Corollary B.4, there exists a node u, such that $u = A(x)$ and $u.ID$ has suffix ω.

First, consider the case where $j = 1$, then $x \in W_{l_1\cdot\omega}$ and $x \notin C_{l_1\cdot\omega}$. Since $u = A(x)$ and $u.ID$ has suffix ω, then it must be that $u \in V_\omega$. However, by Definition 4.8, this implies $x \in C_{l_1\cdot\omega}$. A contradiction. Second, consider the case where $j > 1$. Suppose $u \in C_{l_i...l_1\cdot\omega}$, where $l_i...l_1 \cdot \omega$ is a suffix of both $u.ID$ and $x.ID$. By the definition of $cset(V, W)$, $x \in C_{l_{i+1}...l_1\cdot\omega}$, $l_{j+1} = x[i+k]$, and hence, $x \in C_{l_{i'}...l_1\cdot\omega}$ for all i', $i+1 \leq i' \leq d-k$, where $l_{i'}...l_1 \cdot \omega$ is a suffix of $x.ID$. Therefore, it must be that $i+1 > j$, i.e., $i \geq j$ (otherwise, $x \in C_{l_j...l_1\cdot\omega}$). However, by Corollary B.5, $|C_{l_{j'}...l_1\cdot\omega}| \geq K$ for $1 \leq j' \leq i$, thus, $|C_{l_j...l_1\cdot\omega}| \geq K$. A contradiction. ■

Proposition B.5 *Consider any node x, $x \in W$, if $x \in C_{l_{j+1}...l_1\cdot\omega}$ and $x \notin C_{l_j...l_1\cdot\omega}$, $1 \leq j \leq d-k-1$, (or if $x \in C_{l_1\cdot\omega}$, respectively), then*

(a) there exists a node v, $v \in C_{l_j...l_1\cdot\omega}$ (or $v \in V_\omega$), such that $x \in N_v(j+k, l_{j+1})$ (or $x \in N_v(k, l_1)$) and $A(x) = v$;
(b) $x.att\_level = j + k$ (or $x.att\_level = k$).

Proof: By Corollary B.4, there exists a node u, such that $A(x) = u$. Suppose $u \in C_{l_i...l_1\cdot\omega}$ and $x \in N_u(i+k, x[i+k])$, where $i+k$ is the attach-level of x in $u.table$, $0 \leq i \leq d-k-1$. Hence, $x \in C_{l_{i+1}...l_1\cdot\omega}$, where $l_{i+1} = x[i+k]$ and according to the algorithm, x sets $x.att\_level = i + k$.

Then it must be that $i \geq j$. Otherwise, if $i < j$, then since $x \in C_{l_{i+1}...l_1\cdot\omega}$, it follows that $x \in C_{l_{i'}...l_1\cdot\omega}$, $i' \leq i \leq d-k$, thus $x \in C_{l_j...l_1\cdot\omega}$, which contradicts with the assumption in the proposition.

Next, we show that $i \leq j$, proving by contradiction. Assume $i > j$. Thus $l_i...l_1 \cdot \omega$ is a longer suffix than $l_j...l_1 \cdot \omega$. Since x only sends *JN* to nodes with suffix $x[i+k-1]...x[0]$ (i.e. suffix $l_i...l_1 \cdot \omega$), other nodes can only know x through these nodes plus node u. (Note that x would not be a neighbor at any level lower than level-($i+k$) in tables of these nodes, because when a node, y, copies x, from $z.table$, where z is one of the nodes x has sent *JN* to or $z = u$, if x is stored at levels no lower than level-$i+k$ in $z.table$, then y will not store

x at a level lower than $i+k$. See Figures 10 and 13.) Given that $x \in C_{l_{j+1}...l_1\cdot\omega}$ and $x \notin C_{l_j...l_1\cdot\omega}$, by the definition of $cset(V,W)$, there must exist one node y, $y \in C_{l_j...l_1\cdot\omega}$ and $y \neq x$, such that $x \in N_y(j+k, l_{j+1})$ by time t^e. y can not store x by receiving a *JW* from x, since that indicates $A(x)=y$ and $i=j$, which contradicts with the assumption that $i>j$. Also as discussed above, since $i>j$, x will only send *JN* to nodes with suffix $l_i...l_1\cdot\omega$ and thus will not send a *JN* to y. Hence, y knows x through another node, z. 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. Both cases (i) and (ii) are impossible, because z can only store x at a level no lower than $i+k$ (see Figure 11), thus when y copies x from $z.table$, it can not fill x into a level lower than $i+k$ (again, see Figure 13). 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 $|csuf(x.ID, y.ID)| = j+k$, $|csuf(x.ID, z.ID)| = i+k$ and $i>j$. Therefore, case (iii) is impossible, either. Thus, we conclude that $i \leq j$.

Since $i \geq j$ and $i \leq j$, we conclude that $i=j$. Hence, $u \in C_{l_j...l_1\cdot\omega}$ and $x.att\_level = j+k$, where $u=A(x)$. ■

Corollary B.5 *If $C_{l_j...l_1\cdot\omega}$ is the first C-set x belongs to, $2 \leq j \leq d-k$, then $|C_{l_i...l_1\cdot\omega}| \geq K$ for $1 \leq i < j$.*

Proof: Consider *contact-chain(x,g)* and construct a sequence of nodes, $(g_0, ..., g_h)$, where $h = j+k$, based on *contact-chain(x,g)*, in the same way described in the proof of Proposition 6.1. Thus, $g_j[i'-1]...g_0[0] = x[i'-1]...x[0]$, $0 \leq i' \leq h$. Assume $|C_{l_i...l_1\cdot\omega}| < K$. We know that $g_{k+i} \in C_{l_i...l_1}$. Then, by the definition of *contact-chain(x,g)*, g_{k+1} is a node that x has sent a *CP* or a *JW* to. If $|C_{l_i...l_1\cdot\omega}| < K$, then it must be that $N_{g_{k+i}}(k+i, x[k+i]).size < K$ (implied by Definition 4.8), and hence $N_{g_{k+i}}(h', x[h']).size < K$, where $k+i \leq h' \leq |csuf(x.ID, g_{k+i}.ID)|$. Then x would not send a *CP* to g_{k+1}, since when x finds $N_{g_{k+i}}(k+i, x[k+i]).size < K$, it will change status to *waiting* and send a *JW* to g_{k+1}. However, if x has sent a *JW* to g_{k+i}, then g_{k+i} would store x since an attach-level of x in $g_{k+i}.table$ exists, which $x \in C_{l_i...l_1\cdot\omega}$. A contradiction with that the $C_{l_j...l_1\cdot\omega}$ is the first C-set x belongs to, $j>i$. ■

Proposition B.6 *Consider a node y, $y \in W$, and let $u_y = A(y)$. Suppose $C_{l_j...l_1\cdot\omega}$ is the first C-set y belongs to, $1 \leq j \leq d-k$. Then for a node x, $x \in W$ and $x.ID$ has suffix $l_{j-1}...l_1\cdot\omega$, if $x \xrightarrow{j} u_y$ happens, or $x \in N_{u_y}(j+k-1, l_j)$ before u_y receives the* JW *from y, then by time t_{xy}, $t_{xy} = \max(t^e_x, t^e_y)$, $\langle y \rightarrow x\rangle_d$.*

Proof: Let t_y be the time u_y sends its positive *JWRly* to y, and t_x be the time u_y receives the notification from x if $x \xrightarrow{j} u_y$ happens. Since $u_y = A(y)$, $y \in C_{l_j...l_1\cdot\omega}$ and $y \notin C_{l_{j-1}...l_1\cdot\omega}$, by Proposition B.5, $u_y \in C_{l_{j-1}...l_1\cdot\omega}$ (or $u_y \in V_\omega$ if $j=1$) and $y.att\_level = k+j-1$. Also, we know that before time t_y, $N_{u_y}(k+j-1, l_j).size < K$ (by Fact B.4).

If $x \xrightarrow{j} u_y$ happens and $t_x > t_y$, then x knows y from u_y's reply and $x \xrightarrow{j} y$ will happen. By Proposition B.2, $\langle y \rightarrow x\rangle_d$ by time t^e_x.

If $x \xrightarrow{j} u_y$ happens and $t_x < t_y$, then at time t_x, $N_{u_y}(k+j-1, l_j).size < K$, therefore, u_y stores x into $N_{u_y}(k+j-1, l_j)$. Then, by time t_y, $x \in N_{u_y}(k+j-1, l_j)$. In what follows, we only consider the case that $x \in N_{u_y}(k+j-1, l_j)$ before u_y receives the *JW* from y. In this case, y learns x from u_y's *JWRly*. (i) If y also stores x into $N_y(k+j-1, l_j)$, then trivially, $\langle y \rightarrow x\rangle_d$ by time t^e_y. (ii) Otherwise, $y \xrightarrow{j} x$ eventually happens ($|csuf(x.ID, y.ID)| \geq k+j > y.att\_level$).

(1) If by the time x receives the notification from y, x is still a T-node, then $x \xrightarrow{j} v$ must happen eventually, where $v = N_y(h, x[h]).first$, $h = |csuf(x.ID, y.ID)|$. Thus, $\langle v \rightarrow x\rangle_d$ is by time t^e_x, which implies $\langle y \rightarrow x\rangle_d$ by time t^e_x, since there exists a neighbor sequence $(y, v, v_1, ..., v_f, x)$, where $(v, v_1, ..., v_f, x)$ is the neighbor sequence from v to x.

(2) If by the time x receives the notification from y, x is already an S-node, then x will set a flag to be *true* in its reply to y (see Figure 10). Seeing the flag, y will send a *SN(y,x)* to v, $v = N_y(h, x[h]).first$, $h = |csuf(x.ID, y.ID)|$. v will either store x into $N_v(h', x[h'])$, $h' = |csuf(v.ID, x.ID)|$, or forward *SN(y,x)* to $N_v(h', x[h']).first$), until eventually x is or has been stored by a receiver of the message *SN(y,x)* (see Figure 11) and a *SNRly* is sent back to y. Thus, by time t^e_y, $\langle v \rightarrow x\rangle_d$. Therefore, $\langle y \rightarrow x\rangle_d$ by time t^e_y. ■

Corollary B.6 *If $y \xrightarrow{j} x$ happens, where $x \in W$ and $y \in W$, and $|csuf(x.ID, y.ID)| > y.att\_level$, then $\langle y \rightarrow x\rangle_d$ by time t_{xy}, $t_{xy} = \max(t^e_x, t^e_y)$.*

Proof: See case (2) in the last part of the proof of Proposition B.6. ■

Proposition B.7 *Consider any node x, $x \in V_\omega$. For any C-set, $C_{l\cdot l_{j-1}...l_1\cdot\omega}$, l_1,...,$l_{j-1} \in [b]$ and $l \in [b]$, if $l_{j-1}...l_1\cdot\omega$ is a suffix of $x.ID$, then,*

(a) *for any node y, $y \in C_{l\cdot l_{j-1}...l_1\cdot\omega}$ and $y \in W$, $y \xrightarrow{j} x$ happens before time t^e_y;*

(b) *$N_x(k+j-1, l).size = \min(K, |(V \cup W)_{l\cdot l_{j-1}...l_1\cdot\omega}|)$ holds by time t^e.*

Proof: For any node y, $y \in C_{l\cdot l_{j-1}...l_1\cdot\omega}$, if $y \in W$, then by Proposition B.5, $y.att\_level \leq j+k-1$ and there exists a node u, such that $u = A(y)$. Then $\langle u \rightarrow x\rangle_d$ by the time u sends its *JWRly* to y. (If $u \in V$, then $\langle u \rightarrow x\rangle_d$ because the initial network is consistent; if $u \in W$, then by Corollary B.2, $\langle u \rightarrow x\rangle_d$.) By Proposition B.1, $y \xrightarrow{j} x$ has happened by t^e_y, since $|csuf(x.ID, y.ID)| \geq j-1+k \geq y.att\_level$. Moreover, by Proposition B.2, $\langle x \rightarrow y\rangle_d$ by time t^e_y. Also, by Corollary B.2, $\langle y \rightarrow x\rangle_d$ by time t^e_y. Therefore, part (a) holds.

Since the initial network is K-consistent, we know that before any join happens, $N_x(k+j-1, l) = V_{l\cdot l_{j-1}...l_1\cdot\omega}$ since $|V_{l\cdot l_{j-1}...l_1\cdot\omega}| < K$. Part (a) shows that for any y, $y \in C_{l\cdot l_{j-1}...l_1\cdot\omega}$ and $y \in W$, $y \xrightarrow{j} x$ eventually happens. It then follows that $N_x(k+j-1, l).size = \min(K, |(V \cup W)_{l\cdot l_{j-1}...l_1\cdot\omega}|)$ by time t^e, since by Proposition 6.2, $C_{l\cdot l_{j-1}\ldots l_1\cdot\omega} = (V\cup W)_{l\cdot l_{j-1}\ldots l_1\cdot\omega}$ if $|(V\cup W)_{l\cdot l_{j-1}\ldots l_1\cdot\omega}| < K$, and $|C_{l\cdot l_{j-1}\ldots l_1\cdot\omega}| \geq K$ if $|(V\cup W)_{l\cdot l_{j-1}\ldots l_1\cdot\omega}| \geq K$. ■

Proposition 6.3 *Consider any node x, $x\in V_\omega$. For any C-set $C_{l\cdot l_j\ldots l_1\cdot\omega}$, $0\leq j\leq d-k-1$ and $l\in[b]$, if $l_j\ldots l_1\cdot\omega$ is a suffix of $x.ID$, then $N_x(k+j,l).size = \min(K, |(V\cup W)_{l\cdot l_j\ldots l_1\cdot\omega}|)$ holds by time t^e.*
Proof: By Proposition B.7 (b), the proposition holds. ■

Proposition B.8 *For any C-set, $C_{l_j\ldots l_1\cdot\omega}$, $1\leq j\leq d-k$, $l_1,\ldots,l_j\in[b]$, the following assertions hold:*

(a) If $|W_{l_j\ldots l_1\cdot\omega}|\geq 2$, then for any two nodes, x and y, where $x\in C_{l_j\ldots l_1\cdot\omega}$, $y\in C_{l_j\ldots l_1\cdot\omega}$, $x\neq y$, and x and y are both in W, by time t_{xy}, at least one of $x\xrightarrow{j}y$ and $y\xrightarrow{j}x$ has happened, where $t_{xy}=\max(t^e_x,t^e_y)$. Moreover, at time t_{xy}, $\langle x\rightarrow y\rangle_d$ and $\langle y\rightarrow x\rangle_d$.

(b) For each x, $x\in C_{l_j\ldots l_1\cdot\omega}$ and $x\in W$, $N_x(k+j-1,l).size = \min(K,|(V\cup W)_{l\cdot l_{j-1}\ldots l_1\cdot\omega}|)$ by time t^e, where $l\in[b]$.

Proof: We prove the proposition by induction on j.
Base step: $j=1$. Consider nodes x and y, $x\in W$ and $x\in C_{l_1\cdot\omega}$, $y\in W$ and $y\in C_{l\cdot\omega}$, where $l_1\in[b]$, $l\in[b]$ (l may or may not be the same with l_1), and $x\neq y$. By Proposition B.5, there exists a node u_x, $u_x\in V_\omega$, such that $u_x=A(x)$ (thus, $x\in N_{u_x}(k,l)$). Likewise there exists a node u_y, $u_y\in V_\omega$, such that $y\in N_{u_y}(k,l)$ and $u_y=A(y)$. By Proposition B.5, $x.att\_level = y.att\_level = k$. Therefore, both $x\xrightarrow{j}u_x$ and $y\xrightarrow{j}u_y$ happens. Also, by part(a) of Proposition B.7, $x\xrightarrow{j}u_y$ happens. Likewise, $y\xrightarrow{j}u_x$ happens. By Proposition B.6, $\langle y\rightarrow x\rangle_d$ and $\langle x\rightarrow y\rangle_d$ by time t_{xy}.

Let t_1 be the time u_x sends its reply to x, t_2 be the time u_x sends its reply to y, t_3 be the time u_y sends its reply to y, and t_4 be the time u_y sends its reply to x. Clearly, $t_4>t_1$, because at t_1, x is in status *waiting*, while at t_4, x is in status *notifying*. Likewise, $t_2>t_3$. Note that at time t_1, u_x stores x in $N_{u_x}(k,l)$, and at time t_3, u_y stores y in $N_{u_y}(k,l)$.

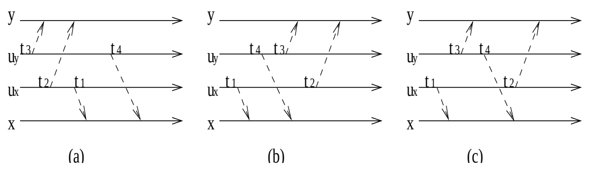


Figure 17: Message sequence chart for base case

If $t_1>t_2$, then it must be $t_4>t_3$, as shown in Figure 17(a). By Fact B.4, $N_{u_x}(k,l).size<K$ before time t_1. Thus, at time t_2, $N_{u_x}(k,l).size<K$. Since $y.ID$ also has suffix $l\cdot\omega$, u_x stores y in $N_{u_x}(k,l)$ at time t_2. Consequently, from u_x's reply, x knows y and stores y in $N_x(k,l)$. (In the copy of $u_x.table$ included in u_x's reply, Since $|csuf(x.ID,y.ID)|\geq k+1$ and $x.att\_level=k$, $x\xrightarrow{j}y$ will happen.

If $t_1<t_2$, then consider the following cases.

- If $t_3>t_4$, as shown in Figure 17(b), then this case is symmetric to the case where $t_1>t_2$, by reversing the role of x and y.
- If $t_3<t_4$, as shown in Figure 17(c), then from u_y's reply, x knows y and will notify y if it has not done so. Similarly, y knows x from u_x's reply and will notify x if it has not done so.

Then, if $l=l_1$, that is, both x and y belong to $C_{l_1\cdot\omega}$, part (a) of the proposition holds, since we have shown above that at least one of $x\xrightarrow{j}y$ and $y\xrightarrow{j}x$ will happen before time t_{xy}, and $\langle x\rightarrow y\rangle_d$ and $\langle y\rightarrow x\rangle_d$ by time t_{xy}.

Part (b) of the proposition also holds, since we have shown above that for any l, $l\in[b]$, $x\xrightarrow{j}y$ or $y\xrightarrow{j}x$ will happen. Thus, eventually x knows y, for each y, $y\in C_{l\cdot\omega}$ and $y\in W$. By Corollary B.1, $N_x(k,l)\supseteq V_{l\cdot\omega}$. Then, eventually, $N_x(k,l).size=\min(K,|(V\cup W)_{l\cdot\omega})|$.

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

Observe that if statement (a) is true, then statement (b) is true if $l=x[k+j-1]$ (i.e. $l=l_j$). The reason is as follows. Statement (a) shows that for any other node in $C_{l_j\ldots l_1\cdot\omega}$, say y, eventually at least one of $x\xrightarrow{j}y$ and $y\xrightarrow{j}x$ happens. Either way, x gets to know y. If x has not stored K neighbors in $N_x(k+j-1,l_j)$ by the time it knows y, it will store y into that entry. By Proposition 6.2, $\min(K,|(V\cup W)_{l_j\ldots l_1\cdot\omega}|)=\min(K,|C_{l\cdot l_j\ldots l_1\cdot\omega}|)$. Thus, by time t^e, either that x has stored K neighbors in $N_x(k+j-1,l_j)$, or it has stored all nodes in $C_{l\cdot l_j\ldots l_1\cdot\omega}$ if the number of nodes in this C-set is less than K.

Based on the above observation, in what follows, when we prove statement (b), we focus on the case where $l\neq l_j$.

Consider node x, $x\in C_{l_{j+1}\ldots l_1\cdot\omega}$ and the following cases:

- **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 prove part(a) of the proposition holds. If $|C_{l_{j+1}\ldots l_1\cdot\omega}|>1$, then consider any node y, $y\in C_{l_{j+1}\ldots l_1\cdot\omega}$, $y\neq x$ and $y\in W$:
    - **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 prove part(b) of the proposition holds. Consider any node y, $y\in C_{l\cdot l_j\ldots l_1\cdot\omega}$, where $l\neq l_i$ and $C_{l\cdot l_j\ldots l_1\cdot\omega}\neq\emptyset$:
    - **1.b.1** $y\notin C_{l_j\ldots l_1\cdot\omega}$.
    - **1.b.2** $y\in C_{l_j\ldots l_1\cdot\omega}$.
- **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, $y\in C_{l_{j+1}\ldots l_1\cdot\omega}$, $y\neq x$ and $y\in W$:
    - **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 any node y, $y\in C_{l\cdot l_j\ldots l_1\cdot\omega}$, where $l\neq l_i$ and $C_{l\cdot l_j\ldots l_1\cdot\omega}\neq\emptyset$:
    - **2.b.1** $y\notin C_{l_j\ldots l_1\cdot\omega}$.
    - **2.b.2** $y\in C_{l_j\ldots l_1\cdot\omega}$.

We will use the following Claim in our proof:

Claim B.2 *Suppose Proposition B.8 holds at j, $1\leq j\leq d-k-1$. If $x\in C_{l_{j+1}\ldots l_1\cdot\omega}$, $y\in C_{l\cdot l_j\ldots l_1\cdot\omega}$, where $l\in[b]$, however,*

$x \notin C_{l_j \ldots l_1 \cdot \omega}$ and $y \notin C_{l_j \ldots l_1 \cdot \omega}$, then either $x \xrightarrow{j} y$ or $y \xrightarrow{j} x$ eventually happens.

Proof of Claim B.2: Observe that the first C-set x belongs to is $C_{l_{j+1} \ldots l_1 \cdot \omega}$, and the first C-set y belongs to is $C_{l \cdot l_j \ldots l_1 \cdot \omega}$. By Proposition B.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), $l \neq l_{j+1}$.

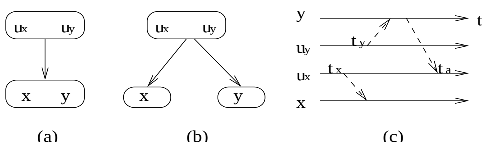


Figure 18: C-sets and message sequences, case 1.a.1 and case 1.b.1

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 B.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 B.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. ■

Case 1.a.1. By Proposition B.5, there exists a node u_x, $u_x \in C_{l_j \ldots l_1 \cdot \omega}$, such that $u_x = A(x)$ and $x.att\_level = j + k$. Likewise, there exists a node u_y, $u_y \in C_{l_j \ldots l_1 \cdot \omega}$, such that $u_y = A(y)$ and $y.att\_level = j + k$. 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 $t_x < t_y$. By Claim B.2, $y \xrightarrow{j} x$ happens. By Proposition B.2, $\langle x \to y \rangle_d$ by time t^e_y.

Next, we need to show $\langle y \to x \rangle_d$ by time t_{xy}. Consider the following cases:

(i) $u_x \in V$ and $u_y \in V$, or $u_x \in W$ and $u_y \in V$. In these two cases, $\langle u_x \to u_y \rangle_d$ by time t_x. By Proposition B.1, $x \xrightarrow{j} u_y$ happens before t^e_x. Then by Proposition B.6, $\langle y \to x \rangle_d$.

(ii) $u_x \in V$ and $u_y \in W$. By Proposition B.7, $u_y \xrightarrow{j} u_x$ happens. Let t_a be the time that u_x receives the notification from u_y.

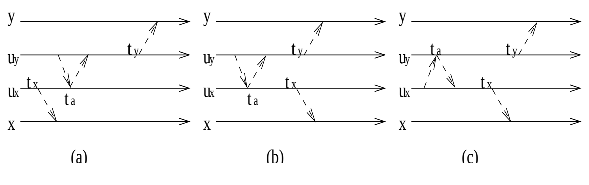


Figure 19: Message sequence chart for case 1.a.1

(1) Suppose $t_x < t_a$, as shown in Figure 19(a). By Fact B.3, $x \in N_{u_x}(l + k, l_{j+1})$ after time t_x. Therefore, when u_x replies to u_y, $x \in u_x.table$. By Facts B.1 and B.2, $t_a < t_y$. By Fact B.4, $N_{u_y}(l + k, l_{j+1}).size < K$ before t_y. Hence, $N_{u_y}(l + k, l_{j+1}).size < K$ at time t_a and therefore, u_y stores x in $N_{u_y}(l+k, l_{j+1})$ at time t_a. By Proposition B.6, $\langle y \to x \rangle_d$.

(2) Suppose $t_x > t_a$, as shown in Figure 19(b). then first consider the case that after u_x receives the notification from u_y, $u_y \in u_x.table$. Then from u_x's reply, x knows u_y and will notify u_y, because $|csuf(u_y.ID, x.ID)| \geq l + k = x.att\_level$ (see Fact B.5). Hence, $x \xrightarrow{j} u_y$ happens. By Proposition B.6, $\langle y \to x \rangle_d$by t_{xy}. Second, consider the case that after u_x receives the notification from u_y, $u_y \notin u_x.table$, then $N_{u_x}(h, u_y[h]).size = K$ at time t_a, $h = |csuf(u_x.ID, u_y.ID)|$. Let $v = N_{u_x}(h, u_y[h]).first$. Then, u_y knows v from u_x's reply Ṡince $u_y.att\_level \leq l - 1 + k$ and $|csuf(v.ID, u_y.ID)| > h \geq l + k$, $u_y \xrightarrow{j} v$ eventually happens. Likewise, x knows v from u_x's reply after time t_x and $x \xrightarrow{j} v$ eventually happens, since $x.att\_level = l + k$ and $|csuf(v.ID, u_y.ID)| \geq l + k$. Then, by Proposition B.4, by time t_{xu_y}, $t_{xu_y} = \max(t^e_x, t^e_{u_y})$, either that $x \in N_{u_y}(l + k, l_{j+1})$ or $N_{u_y}(l + k, l_{j+1}) = K$. $N_{u_y}(l + k, l_{j+1}) = K$ is impossible, because $N_{u_y}(l+k, l_{j+1}) < K$ before time t_y, and $t_y > t_{xu_y}$ (we have assumed $t_y > t_x$, and $t_y \geq t^e_{u_y}$ by Fact B.2). Thus, $x \in N_{u_y}(l + k, l_{j+1})$ at time t_{xu_y}. By Proposition B.6, $\langle y \to x \rangle_d$ by t_{xy}.

(iii) $u_x \in W$ and $u_y \in W$. Then, by assuming the proposition holds at j, either $u_y \xrightarrow{j} u_x$ or $u_x \xrightarrow{j} u_y$ happens.

(1) If $u_y \xrightarrow{j} u_x$ happens and $t_x < t_a$, then following the same arguments in part (1) of the above case (ii) ($u_x \in V$ and $u_y \in W$), $\langle y \to x \rangle_d$ by t_{xy}.

(2) If $u_y \xrightarrow{j} u_x$ happens and $t_x > t_a$, then following the same arguments in part (2) of the above case (ii) ($u_x \in V$ and $u_y \in W$), $\langle y \to x \rangle_d$ by t_{xy}.

(3) If $u_x \xrightarrow{j} u_y$ happens, let t_a be the time u_x sends its notification to u_y, then by Facts B.1 and B.2, it must be $t_x > t_a$, as shown in Figure 19(c). At time t_a, u_x already knows u_y. Then, there are two cases to consider: $u_y \in u_x.table$ or $u_y \notin u_x.table$ at time t_x. Following the same argument as in part (2) of case (ii), it can be proved that $\langle y \to x \rangle_d$.

Case 1.a.2 First, observe that in this case, $y.att\_level \leq j + k - 1 < |csuf(y.ID, x.ID)|$. Let $u_x = A(x)$, then $u_x \in C_{l_j \ldots l_1 \cdot \omega}$. Thus both u_x and y belong to $C_{l_j \ldots l_1 \cdot \omega}$, as shown in Figure 20(a). If $u_x \in V$, then by Proposition B.7, $y \xrightarrow{j} u_x$ happens by t^e_y. If $u_x \in W$, by assuming the proposition holds at j, we know that by the time both u_x and y are S-nodes, they can reach each other; moreover, at least one of $y \xrightarrow{j} u_x$ and $u_x \xrightarrow{j} y$ happens.

Let t_1 be the time u_x sends its *JWRly* to x. Also, let 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.

(i) If $t_1 < t_2$, then at t_2, $x \in N_{u_x}(k + j, l_{j+1})$. Then t_2 must be the time that u_x receives the notification from y (by Fact B.2, at time t_2 u_x is already an S-node and will

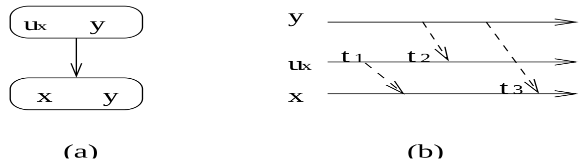


Figure 20: Message sequence chart for case 1.a.2

not send out notifications), as shown in Figure 20(b) . Thus y knows x from u_x's reply that includes $u_x.table$, and will notify x if it has not done so. Thus, $y \xrightarrow{j} x$ happens by time t^e_y. By Proposition B.2, $\langle x \rightarrow y\rangle_d$. Also, since $y \xrightarrow{j} x$ happens, and $|csuf(x.ID, y.ID)| \geq k+j+1 > y.att\_level$, by Corollary B.6, $\langle y \rightarrow x\rangle_d$ by t_{xy}.

(ii) If $t_1 > t_2$ and $y \xrightarrow{j} u_x$ happens, then it must be that $y \in N_{u_x}(l+k, l_{j+1})$ after time t_2. By Fact B.4, $N_{u_x}(l+k, l_{j+1}).size < K$ before t_1, thus $N_{u_x}(l+k, l_{j+1}).size < K$ before t_2. Then, by Proposition B.6, $\langle x \rightarrow y\rangle_d$. Moreover, $x \xrightarrow{j} y$ happens, because $x.att\_level = j+k$ and $|csuf(x.ID, y.ID)| \leq j+k$. By Proposition B.2, $\langle y \rightarrow x\rangle_d$.

(iii) If $t_1 > t_2$ and $u_x \xrightarrow{j} y$ happens, then following the same argument above in case (ii), it must be that $y \in N_{u_x}(l+k, l_{j+1})$ after time t_2, and therefore, $\langle x \rightarrow y\rangle_d$ and $\langle y \rightarrow x\rangle_d$. Moreover, $x \xrightarrow{j} y$ happens.

Case 2.a.1 This case is symmetric to case 1.a.2.

Case 2.a.2 In this case, both x and y also belong to $C_{l_j \ldots l_1 \cdot \omega}$. By assuming Proposition B.8 holds at j, part(a) holds in case 2.a.2 trivially.

So far, we have proved that part (a) of Proposition B.8 holds. Next, we prove part (b). As we mentioned before, here we focus on the case where $l \neq l_j$.

Case 1.b Consider node y, $y \in C_{l \cdot l_j \ldots l_1 \cdot \omega}$. If $y \in V$, then by Corollary B.1, $y \in N_x(j+k, l)$ by time t^e. Hence, in what follows, we only consider the case where $y \in C_{l \cdot l_j \ldots l_1 \cdot \omega}$ and $y \in W$. (1) If $y \notin C_{l_j \ldots l_1 \cdot \omega}$ (Case 1.b.1), then by Claim B.2, either $x \xrightarrow{j} y$ or $y \xrightarrow{j} x$ eventually happens. In either case, x eventually knows y. Therefore, either $y \in N_x(j+k, l)$ or $N_x(j+k, l).size = K$ at the time x knows y. (2) If $y \in C_{l_j \ldots l_1 \cdot \omega}$ (Case 1.b.2), then by assuming the proposition holds at j, we have $y \xrightarrow{j} u_x$ or $u_x \xrightarrow{j} y$ happens if $u_x \in W$; and $y \xrightarrow{j} u_x$ happens if $u_x \in V$, by Proposition B.7. Let t_x be the time u_x sends its positive *JWRly* to x. Let t_a be the time u_x receives the notification from y if $y \xrightarrow{j} u_x$ happens; otherwise, let t_a be the time u_x sends a notification to y.

- If $y \xrightarrow{j} u_x$ happens and $t_x < t_a$ (then t_a is the time u_x receives the notification from y), then y knows x from u_x's reply and $y \xrightarrow{j} x$ happens.
- If $y \xrightarrow{j} u_x$ happens and $t_x > t_a$, then either $y \in N_{u_x}(j+k, l)$ or $N_{u_x}(j+k, l) = K$ at time t_x, and therefore, either $y \in N_x(j+k, l)$ or $N_x(j+k, l).size = K$ after x receives u_x's reply (*JWRly*) and copies nodes from $u_x.table$.
- If $u_x \xrightarrow{j} y$ happens, then $t_x > t_a$. Similar to the above argument, either $y \in N_x(j+k, l)$ or $N_x(j+k, l) = K$ after x receives u_x's reply and copies nodes from $u_x.table$.

The above analysis shows that for each node y, $y \in C_{l \cdot l_j \ldots l_1 \cdot \omega}$, either that after time t_x, $y \in N_x(j+k, l)$, $N_x(j+k, l) = K$, or x eventually is notified by y. By Proposition 6.2, $|C_{l \cdot l_j \ldots l_1 \cdot \omega}| = \min(K, |(V \cup W)_{l \cdot l_j \ldots l_1 \cdot \omega}|)$. Hence, $N_x(j+k, l).size = \min(K, |(V \cup W)_{l \cdot l_j \ldots l_1 \cdot \omega}|)$.

Case 2.b Consider node y, $y \in C_{l \cdot l_j \ldots l_1 \cdot \omega}$. Again, we only consider the case where $y \in W$ (if $y \in V$, by Corollary B.1, $y \in N_x(j+k, l)$ by time t^e). (i) If $y \in C_{l_j \ldots l_1 \cdot \omega}$, then both x and y belong to $C_{l_j \ldots l_1 \cdot \omega}$. By assuming the proposition holds at j, at least one of $x \xrightarrow{j} y$ or $y \xrightarrow{j} x$ happens. Hence, x eventually knows y. (ii) If $y \notin C_{l_j \ldots l_1 \cdot \omega}$, then $A(y) \in C_{l_j \ldots l_1 \cdot \omega}$. Let $u_y = A(y)$, and t_y be the time u_y sends its positive *JWRly* to y. Recall that in this case, both x and u_y belong to $C_{l_j \ldots l_1 \cdot \omega}$. If $u_y \in W$, then by assuming the proposition holds at j, at least one of $x \xrightarrow{j} u_y$ or $u_y \xrightarrow{j} x$ happens; if $u_y \in V$, then by Proposition B.7, $x \xrightarrow{j} u_y$ happens. Let t_a be the time u_y sends its notification to x if $u_y \xrightarrow{j} x$ happens; otherwise, let t_a be the time u_y receives the notification from x. If $t_a < t_y$, then by time t_a, u_y already knows x. Then by time t_y, $N_{u_y}(j+k, l).size < K$, and thus at time t_a, $N_{u_y}(j+k, l).size < K$. Hence, u_y will store x into $N_{u_y}(j+k, l)$ at time t_a. Hence, at time t_y, $x \in N_{u_y}(j+k, l)$. Then, from u_y's reply, y knows x and will send a *JN* to x ($y \xrightarrow{j} x$), which enables x to know the existence of y. If $t_a > t_y$, then at time t_a, $y \in N_{u_y}(j+k, l)$. Hence, from u_y'reply (or u_y's notification), x knows the existence of y. So far, we have shown that whether $y \in C_{l_j \ldots l_1 \cdot \omega}$ or not, x eventually knows y. This is true for any y, $y \in C_{l \cdot l_j \ldots l_1 \cdot \omega}$. By Proposition 6.2 and Corollary B.1, $N_x(j+k, l).size = \min(K, |(V \cup W)_{l \cdot l_j \ldots l_1 \cdot \omega}|)$. Therefore, part (b) of the proposition holds in Case 2.b. ■

Corollary B.7 *If $x \in C_{l_j \ldots l_1 \cdot \omega}$ and $C_{l \cdot l_{j-1} \ldots l_1 \cdot \omega} \neq \emptyset$, $l \in [b]$, then for any node y, $y \in C_{l \cdot l_{j-1} \ldots l_1 \cdot \omega}$ and $y \neq x$, at least one of the following assertions is true:*

1. *$y \xrightarrow{j} x$ has happened by time t^e;*
2. *By time t^e_x, either $y \in N_x(j-1+k, l)$ or $N_x(j-1+k, l).size = K$ holds.*

Proof: Proof of the corollary is implied by the proof of Proposition B.8. If $x \in V$, then by Proposition B.7, the corollary holds. If $x \in W$ and $y \in V$, then by Corollary B.1, $y \in N_x(j-1+k, l)$, hence, the corollary also holds. In what follows, we consider the case where $x \in W$ and $y \in W$.

First, suppose $j = 1$. Consider a node x, $x \in C_{l_1 \cdot \omega}$, $l_1 = x[k]$. In the proof of base case in Proposition B.8, we have shown that for any node y, and $y \in C_{l \cdot \omega} \neq \emptyset$, $l \in [b]$, at least one of $y \xrightarrow{j} x$ or $x \xrightarrow{j} y$ happens eventually. If $y \xrightarrow{j} x$ happens, the the proposition holds. Otherwise, if only $x \xrightarrow{j} y$ happens, then x knows y before t^e_x. Hence, either $y \in N_x(k, l)$ or $N_x(k, l).size = K$.

Second, suppose $1 < j \leq d-k$. Consider a node x, $x \in C_{l_j \ldots l_1 \cdot \omega}$, there are following cases:

- $x \notin C_{l_{j-1} \ldots l_1 \cdot \omega}$. Consider any node y, $y \in C_{l \cdot l_{j-1} \ldots l_1 \cdot \omega}$. First, suppose $y \notin C_{l_{j-1} \ldots l_1 \cdot \omega}$. By Claim B.2, at least one of $y \xrightarrow{j} x$ and $x \xrightarrow{j} y$ happens eventually. If $y \xrightarrow{j} x$ happens, then the proposition holds. Otherwise, if only $x \xrightarrow{j} y$ happens, then x knows y before t^e_x. Hence, either $y \in N_x(j-1+k, l)$ or $N_x(j-1+k,l).size = K$ by t^e_x. Second, suppose $y \in C_{l_{j-1}...l_1 \cdot \omega}$. By the proof of Case 1.b in proving Proposition B.8, either $y \xrightarrow{j} x$ eventually happens, or that $y \in N_x(j+k, l)$ or $N_x(j+k,l) = K$ after x receives u_x's reply (*JWRly*) and copies nodes from $u_x.table$, where $u_x = A(x)$.

- $x \in C_{l_{j-1}...l_1 \cdot \omega}$. Again, consider any node y, $y \in C_{l \cdot l_{j-1}...l_1 \cdot \omega}$. First, suppose $y \in C_{l_{j-1}...l_1 \cdot \omega}$, then both x and y belong to $C_{l_{j-1}...l_1 \cdot \omega}$. By part(a) of Proposition B.8, at least one of $x \xrightarrow{j} y$ or $y \xrightarrow{j} x$ happens eventually. Similar to the argument above, at least one of the following is true:$y \xrightarrow{j} x$, $y \in N_x(j-1+k,l)$ or $N_x(j-1+k,l).size = K$.
Second, suppose $y \notin C_{l \cdot l_{j-1}...l_1 \cdot \omega}$. By the proof of Case 2.b in proving Proposition B.8, either $y \xrightarrow{j} x$ eventually happens, or that $y \in N_x(j+k,l)$ or $N_x(j+k,l) = K$ after x receives u_y's reply (or notification) and copies nodes from $u_x.table$, where $u_y = A(y)$. ■

Proposition 6.4 *For any C-set, $C_{l_j...l_1 \cdot \omega}$, $1 \le j \le d-k$, l_1,...,$l_j \in [b]$, the following assertion holds by time t^e: For each x, $x \in C_{l_j...l_1 \cdot \omega}$ and $x \in W$, $N_x(k+j-1,l).size = \min(K, |(V \cup W)_{l \cdot l_{j-1}...l_1 \cdot \omega}|)$, $l \in [b]$.*

Proof: By Proposition B.8(b), the proposition holds. ■

Proposition 6.5 *For any x, $x \in W$, suppose $C_{l_j...l_1 \cdot \omega}$ is the first C-set x belongs to, where $l_j...l_1 \cdot \omega$ is a suffix of $x.ID$, $1 \le j \le d-k$. Then for any i, $0 \le i \le j$ and any l, $l \in [b]$, $N_x(k+i,l).size = \min(K, |(V \cup W)_{l \cdot l_i...l_1 \cdot \omega}|)$.*

Proof: Consider *contact-chain(x,g)*, where g is the node that x is given to start its join process. Suppose *contact-chain(x,g)* is $(u_0, u_1, ...u_f, u_{f+1})$, where $u_0 = g$, u_f is the node that sends an positive *JWRly* to x (see Definition of a *contact-chain*, in [8]) and $u_{f+1} = x$. T the lowest level u_f stores x in $u_f.table$ (the attach-level of x) is level-j (by Proposition B.5), then $k \le j \le |csuf(u.ID, x.ID)|$ (recall k is defined in Assumption 1). Create a new sequence $(g_0, ..., g_j)$ as described in the proof of Proposition 6.1, such that $g_0 = g$, $g_j = u_f$, and $g_{i'}.ID$ shares suffix $x[i'-1]...x[0]$ with $x.ID$, $0 \le i' \le j$. Then, it is easy to check that $g_k \in V_\omega$, and $g_{i'+k} \in C_{l_{i'}...l_1 \cdot \omega}$, $1 \le i' \le j$. Thus, $g_{i+k} \in C_{l_i...l_1 \cdot \omega}$. Since g_{i+k} is a node in *contact-chain(x,g)*, either $x \xrightarrow{c} g_{i+k}$ or $x \xrightarrow{j} g_{i+k}$ happens. No matter which happens, let the time g_{i+k} sends the reply to x be t_1.

If $|(V \cup W)_{l \cdot l_i...l_1 \cdot \omega}| < K$, then by Proposition 6.2, $C_{l \cdot l_i...l_1 \cdot \omega} = (V \cup W)_{l \cdot l_i...l_1 \cdot \omega}$, i.e., for each y, $y \in W_{l \cdot l_i...l_1 \cdot \omega}$, $y \in C_{l \cdot l_i...l_1 \cdot \omega}$. Next, consider any node y, $y \in W_{l \cdot l_i...l_1 \cdot \omega}$. Then, if $g_{i+k} \in W$, by Corollary B.7, at least one of the following is true: $y \in N_{g_{i+k}}(i-1+k,l)$ by time t_1 ($t_1 > t^e_{g_{i+k}}$), or that $y \xrightarrow{j} g_{i+k}$ happens by time t^e_y; if $g_{i+k} \in V$, then $y \xrightarrow{j} g_{i+k}$ eventually happens by Proposition B.7. (i) If $y \in N_{g_{i+k}}(i-1+k,l)$ by time t_1, then x knows y from g_{i+k}'s reply, hence $y \in N_x(i-1+k,l)$ or $N_x(i-1+k,l).size$ after x receives the reply from g_{i+k}. (ii) If $y \xrightarrow{j} g_{i+k}$ happens, then by Proposition B.4, at least one of the following is true: by time t^e_x, $y \in N_x(i-1+k,l)$, or that $N_x(i-1+k,l).size = K$. Since this conclusion is true for each y, $y \in C_{l \cdot l_i...l_1 \cdot \omega}$, plus that $V_{l \cdot l_i...l_1 \cdot \omega} \subset N_x(i-1+k,l)$ by time t^e (by Corollary B.1), we conclude that $N_x(i-1+k,l).size = \min(K, |(V \cup W)_{l \cdot l_i...l_1 \cdot \omega}|)$ by time t^e.

If $|(V \cup W)_{l \cdot l_i...l_1 \cdot \omega}| \ge K$, then by Proposition 6.2, $|C_{l \cdot l_i...l_1 \cdot \omega}| \ge K$. Next, consider any node y, $y \in C_{l \cdot l_i...l_1 \cdot \omega}$ and $y \in W$. Let the time g_{i+k} receives the message (either a *CP* or a *JW*) from x be t_1. Then, by Corollary B.7, at least one of the following is true: $y \in N_{g_{i+k}}(i-1+k,l)$ by time t_1, or $N_{g_{i+k}}(i-1+k,l).size = K$ by time t_1, or that $y \xrightarrow{j} g_{i+k}$ happens. (i) If at time t_1, $N_{g_{i+k}}(i-1+k,l).size = K$, then $N_x(i-1+k,l).size = K$. (ii) If at time t_1, $N_{g_{i+k}}(i-1+k,l).size < K$ and $y \in N_{g_{i+k}}(i-1+k,l)$, then $y \in N_x(i-1+k,l)$ or $N_x(i-1+k,l).size = K$ after x receives the reply from g_{i+k}. (iii) If $y \xrightarrow{j} g_{i+k}$ happens, then by Proposition B.4, by time t^e_x, either $y \in N_x(i-1+k,l)$ or $N_x(i-1+k,l).size = K$. Therefore, for any y, $y \in C_{l \cdot l_i...l_1 \cdot \omega}$, either that $y \in N_x(i-1+k,l)$ by time t^e_x, or $N_x(i-1+k,l).size = K$ by time t^e_x. Hence, $N_x(i-1+k,l).size = K$ by time t^e. ■

Proposition 6.6 *For any node x, $x \in W$, if $(V \cup W)_{l \cdot l_j...l_1 \cdot \omega} \neq \emptyset$, where $l_j...l_1 \cdot \omega$ is a suffix of $x.ID$, $0 \le j \le d-k-1$, and $l \in [b]$, then $N_x(k+j,l).size = \min(K, |(V \cup W)_{l \cdot l_j...l_1 \cdot \omega}|)$ holds by time t^e.*

Proof: If $(V \cup W)_{l \cdot l_i...l_1 \cdot \omega} = V_{l \cdot l_i...l_1 \cdot \omega}$, by Corollary B.1, the proposition holds. If $(V \cup W)_{l \cdot l_i...l_1 \cdot \omega} \supset V_{l \cdot l_i...l_1 \cdot \omega}$, then consider C-set $C_{l_j...l_1 \cdot \omega}$. Suppose $C_{l_j...l_1 \cdot \omega}$ is the first C-set x belongs to, $0 \le j \le d-k$. If $j > i$, by Proposition 6.5, the proposition holds. If $j \le i$, then by part(b) of Proposition B.8, the proposition holds. ■

Proposition 6.7 *For each node x, $x \in V \cup W$, $N_x(i+k,j).size = \min(K, |(V \cup W)_{j \cdot x[i-1]...x[0]}|)$ holds by time t^e, $i \in [d]$, $j \in [b]$.*
Proof: First, pick any node x, $x \in W$.

- If $0 \le i < k$, then by Corollary B.1, the proposition holds.
- If $i = k$ and $|V_{j \cdot x[i-1]...x[0]}| \ge K$, then again by Corollary B.1, the proposition holds.
- If $i = k$, $|V_{j \cdot x[i-1]...x[0]}| < K$, however, $W_{j \cdot x[i-1]...x[0]} \neq \emptyset$, or $k < i \le d-1$, then by Proposition 6.6, the proposition holds.

Second, consider nodes in V. Pick y, $y \in V$.

- If $(V \cup W)_{j \cdot y[i-1]...y[0]} = V_{j \cdot y[i-1]...y[0]}$, then given that $\langle V, \mathcal{N}(V) \rangle$ is a K-consistent network, $N_y(i+k,l).size = \min(K, |V_{j \cdot y[i-1]...y[0]}|) = \min(K, |(V \cup W)_{j \cdot y[i-1]...y[0]}|)$. The proposition holds.
- If $V_{j \cdot y[i-1]...y[0]} \subset (V \cup W)_{j \cdot y[i-1]...y[0]}$, then ω must be a suffix of $j \cdot y[i-1]...y[0]$, which can be deduced from Assumption 1 ($V^{Notify}_x = V_\omega$ for any x, $x \in W$), thus $y \in V_\omega$. If $\omega = j \cdot y[i-1]...y[0]$, then $V_\omega = V_{j \cdot y[i-1]...y[0]}$, and $|V_\omega| \ge K$ by Assumption 1. Thus $N_y(i+k,l).size = K$. If $\omega \neq j \cdot y[i-1]...y[0]$, then ω must be shorter than

$j \cdot y[i-1]...y[0]$. By part (b) of Proposition B.7, $N_y(i+k, l).size = \min(K, |(V \cup W)_{j \cdot y[i-1]...y[0]}|)$ by time t^e. The proposition holds. ■

Propositions 6.1 to 6.7 are based on the assumption that all joining nodes belong to the same C-set tree. Next, we consider the case where the joining nodes belong to different C-set trees.

Proposition 6.8 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, join a K-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 $\omega_1 \neq \omega_2$ and ω_2 is a suffix of ω_1. Let $k_2 = |\omega_2|$. Then, by time t^e, for any x, $x \in G(V_{\omega_1})$, the following assertion holds: $N_x(k_2, l).size = \min(K, |(V \cup W)_{l \cdot \omega_2}|)$, $l \in [b]$.*

Proof:

(i) If $|V_{l\cdot\omega_2}| \geq K$, then $N_x(k_2, l).size = K$ by Corollary B.1.

(ii) If $|V_{l\cdot\omega_2}| < K$ and $W_{l\cdot\omega_2} = \emptyset$ then $N_x(k_2, l).size = V_{l\cdot\omega_2}$ by Corollary B.1.

(iii) If $|V_{l\cdot\omega_2}| < K$ and $W_{l\cdot\omega_2} \neq \emptyset$, then it must be that $W_{l\cdot\omega_2} = G(V_{\omega_2})_{l\cdot\omega_2}$, that is, the set of nodes in W with suffix $l \cdot \omega_2$ are the same set of nodes in $G(V_{\omega_2})$ with suffix $l \cdot \omega_2$. We prove $W_{l\cdot\omega_2} = G(V_{\omega_2})_{l\cdot\omega_2}$ by contradiction. Suppose there exists a node z, $z \in W_{l\cdot\omega_2}$, however, $z \in G(V_{\omega_3})$, i.e., $V_z^{Notify} = V_{\omega_3}$, where $\omega_3 \neq \omega_2$. Then, by the definition of V_z^{Notify}, $|V_{\omega_3}| \geq K$ and $|V_{z[k_3]\cdot\omega_3}| < K$, where $k_3 = |\omega_3|$. Since $|V_{l\cdot\omega_2}| < K$, and both $l \cdot \omega_2$ and ω_3 are suffixes of $z.ID$, then ω_3 must be a suffix of ω_2 (if $l \cdot \omega_2$ is a suffix of ω_3, then $V_{l\cdot\omega_2} \supseteq V_{\omega_3}$, and thus $|V_{l\cdot\omega_2}| \geq |V_{\omega_3}| \geq K$, which contradicts with $|V_{l\cdot\omega_2}| \leq |V_{\omega_2}| < K$). And since $\omega_2 \neq \omega_3$, $|\omega_2| > |\omega_3|$. Hence, $z[k_3] \cdot \omega_3$ is a suffix of ω_2 since both of them are suffixes of $z.ID$. Thus, $V_{z[k_3]\cdot\omega_3} \supseteq V_{\omega_2}$, thus $|V_{z[k_3]\cdot\omega_3}| \geq |V_{\omega_2}| \geq K$, which contradicts with $|V_{z[k_3]\cdot\omega_3}| < K$ (by assuming $V_z^{Notify} = V_{\omega_3}$).

For any x, $x \in G(V_{\omega_1})$, consider *contain-chain(x,g)*, where g is the node x is given to start joining, and create a sequence of nodes $g_0, g_1, ..., g_h$ following the same way as discussed in the proof of Proposition 6.1, where $g_0 = g$, $g_h = A(u)$, and g_i shares one more digit with x than g_{i-1}, $1 \leq i \leq h$. Clearly, $k_2 < k_1 \leq h$. Then, g_{k_2} has suffix ω_2 and thus $g_{k_2} \in V_{\omega_2}$. Also, $x \xrightarrow{c} g_{k_2}$ or $x \xrightarrow{j} g_{k_2}$ happens.

Next, we show that there exists a node in $G(V_{\omega_2})$ such that it eventually notifies g_{k_2}. Consider any node v, $v \in C_{l\cdot\omega_2}$ and $v \in W$ (by Proposition 6.2 , such a node must exist). By Proposition B.5, there exists a node u_v, such that $u_v = A(x)$ and $u_v \in V_{\omega_2}$. Hence, $v \xrightarrow{j} u_v$ happens. By Proposition B.1, $v \xrightarrow{j} u$ eventually happens for each u, $u \in V_{\omega_2}$ (by the time u_v replies to v, $\langle u_v \rightarrow u\rangle_d$ already holds since the initial network is consistent). Since $g_{k_2} \in V_{\omega_2}$, we know $v \xrightarrow{j} g_{k_2}$ eventually happens.

Then, by Proposition B.4, by time t^e, either $v \in N_x(k_2, l)$ or $N_x(k_2, l).size = K$ is true. This conclusion is true for each v, $v \in C_{l\cdot\omega_2}$ and $v \in W$. That is, $N_x(k_2, l).size = \min(K, |C_{l\cdot\omega_2}|)$. By Proposition 6.2, $\min(K, |C_{l\cdot\omega_2}) = |\min(K, |(V \cup W)_{l\cdot\omega_2}|$. Therefore, by time t^e, $N_x(k_2, l).size = \min(K, |(V \cup W)_{l\cdot\omega_2}|)$. ■

With the above propositions, we now can prove Lemma 6.4.

Proof of Lemma 6.4: 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, $x \in G(V_{\omega_i})$ if and only if $V_x^{Notify} = V_{\omega_i}$, $1 \leq i \leq h$. Let $\Omega = \{\omega_i, 1 \leq i \leq h\}$. Then, at time t^e,

- Consider a node x, $x \in V$. If $|V_{j\cdot x[i-1]...x[0]}| \geq K$, then $N_x(i,j).size = K$ since initially $\langle V, \mathcal{N}(V)\rangle$ is K-consistent. If $|V_{j\cdot x[i-1]...x[0]}| < K$ and $W_{j\cdot x[i-1]...x[0]} = \emptyset$, then $N_x(i,j).size = |V_{j\cdot x[i-1]...x[0]}| = |(V \cup W)_{j\cdot x[i-1]...x[0]}|$.
  If $|V_{j\cdot x[i-1]...x[0]}| < K$ and $W_{j\cdot x[i-1]...x[0]} \neq \emptyset$, then $j \cdot x[i-1]...x[0] \notin \Omega$, because we know that for any $\omega, \omega \in \Omega$, $|V_\omega| \geq K$ by Definition 4.4. Also, we know that there must exist a ω_l, $\omega_l \in \Omega$, such that ω_l is a suffix of $j \cdot x[i-1]...x[0]$, since $W = \cup_{l=1}^{h} G(V_{\omega_l})$ and any node in $G(V_{\omega_l})$ has suffix ω_l, $\omega_l \in \Omega$.

  Claim B.3 *Suppose $|V_{j\cdot x[i-1]...x[0]}| < K$ and $W_{j\cdot x[i-1]...x[0]} \neq \emptyset$. Also suppose there exists a ω_l, such that $\omega_l \in \Omega$, ω_l is a suffix of $j \cdot x[i-1]...x[0]$, and $|\omega_l| \geq |\omega_h|$ for any ω_h, $\omega_h \in \Omega$ and ω_h is a suffix of $j \cdot x[i-1]...x[0]$. Then, $W_{j\cdot x[i-1]...x[0]} = G(V_{\omega_l})_{j\cdot x[i-1]...x[0]}$.*

  Proof of Claim B.3: Clearly, $G(V_{\omega_l})_{j\cdot x[i-1]...x[0]} \subseteq W_{j\cdot x[i-1]...x[0]}$. We only need to show $W_{j\cdot x[i-1]...x[0]} \subseteq G(V_{\omega_l})_{j\cdot x[i-1]...x[0]}$. In other words, we need to show that for any node y, $y \in W_{j\cdot x[i-1]...x[0]}$, $V_y^{Notify} = V_{\omega_l}$ (thus $y \in G(V_{\omega_l})_{j\cdot x[i-1]...x[0]}$).
  For any node y, $y \in W_{j\cdot x[i-1]...x[0]}$, $j \cdot x[i-1]...x[0]$ is a suffix of $y.ID$. Since ω_l is a suffix of $j \cdot x[i-1]...x[0]$ and $\omega_l \neq j \cdot x[i-1]...x[0]$, ω_l is also a suffix of $y.ID$. By the definition of $G(V_{\omega_l})$, we know that $|V_{\omega_l}| \geq K$. In order to prove $V_y^{Notify} = V_{\omega_l}$, we need to show that $|V_{y[k_l]\cdot\omega_l}| < K$, where $k_l = |\omega_l|$. We prove it by contradiction. Assume $|V_{y[k_l]\cdot\omega_l}| \geq K$, then $V_y^{Notify} = V_{\omega_y}$, where $y[k_l] \cdot \omega_l$ is a suffix of ω_y. Hence, ω_l is a suffix of ω_y and $\omega_l \neq \omega_y$. Since $y \in W$, $\omega_y \in \Omega$. On the other hand, ω_y must be a suffix of $j \cdot x[i-1]...x[0]$, since it is given $|V_{j\cdot x[i-1]...x[0]}| < K$. However, ω_l is picked in such a way that for any ω_h, such that $\omega_h \in \Omega$ and ω_h is also a suffix of $j \cdot x[i-1]...x[0]$, $|\omega_l| \geq |\omega_h|$. Therefore, ω_y must be a suffix of ω_l, which contradicts with the above conclusion: ω_l is a suffix of ω_y and $\omega_l \neq \omega_y$. ■
  By part (b) of Proposition B.7 $N_x(i,j).size = \min(K, |(V \cup G(V_{\omega_l}))_{j\cdot x[i-1]...x[0]}|)$. Then, by Claim B.3, $N_x(i,j).size = \min(K, |(V \cup W)_{j\cdot x[i-1]...x[0]}|)$ by time t^e.
- Consider a node x, $x \in W$. Then there exists a f, $1 \leq f \leq h$, such that $x \in G(V_{\omega_f})$. (i) If $|V_{j\cdot x[i-1]...x[0]}| \geq K$, then $N_x(i,j).size = K$ by Corollary B.1. (ii) If $|V_{j\cdot x[i-1]...x[0]}| < K$ and $W_{j\cdot x[i-1]...x[0]} = \emptyset$, then $N_x(i,j).size = |V_{j\cdot x[i-1]...x[0]}| = |(V \cup W)_{j\cdot x[i-1]...x[0]}|$, again, by Corollary B.1.
  (iii) If $|V_{j\cdot x[i-1]...x[0]}| < K$ and $W_{j\cdot x[i-1]...x[0]} \neq \emptyset$, then $j \cdot x[i-1]...x[0] \notin \Omega$. Since both ω_f and $x[i-1]...x[0]$ are suffixes of $x.ID$, we next consider two cases: ω_f is a suffix of $x[i-1]...x[0]$ or vice versa. If ω_f is a suffix of $x[i-1]...x[0]$, then for any node y, $y \in W_{j\cdot x[i-1]...x[0]}$, $y \in G(V_{\omega_f})$ (that is, x and y are in the same C-set tree). By Proposition 6.6, $N_x(i,j).size = \min(K, (V \cup$

$G(V_{\omega_f}))_{j\cdot x[i-1]...x[0]})$, thus $N_x(i,j).size = \min(K, (V \cup W)_{j\cdot x[i-1]...x[0]})$.

If $x[i-1]...x[0]$ is a suffix of ω_f, then there must exist a ω_l, $\omega_l \in \Omega$ and $\omega_l \neq \omega_f$, such that ω_l is the longest suffix of $j \cdot x[i-1]...x[0]$ among Ω. Then, by Claim B.3, for any node y, $y \in W_{j\cdot x[i-1]...x[0]}$, $y \in G(V_{\omega_l})$ (x and y are in different C-set trees). Note that since $|V_{j\cdot x[i-1]...x[0]}| < K$ and $|V_{\omega_l}| \geq K$, it is impossible that $j \cdot x[i-1]...x[0] = \omega_l$. Hence, ω_l is a suffix of $x[i-1]...x[0]$, which is a suffix ω_f. Therefore, ω_l is a suffix of ω_f, then by Proposition 6.8, $N_x(i,j).size = \min(K, (V \cup G(V_{\omega_l}))_{j\cdot x[i-1]...x[0]})$, thus $N_x(i,j).size = \min(K, (V \cup W)_{j\cdot x[i-1]...x[0]})$. ■

Lemma 6.5 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, join a K-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 a K-consistent network.*

Proof of Lemma 6.5: 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 $G_1 = \emptyset$):

1. Pick any node x, $x \in W - \bigcup_{j=1}^{i-1} G_j$, and put x in G_i.
2. For each node y, $y \in W - \bigcup_{j=1}^{i} G_j$,
   (a) if there exists a node z, $z \in G_i$, such that $(V_y^{Notify} \cap V_z^{Notify} \neq \emptyset)$, then put y in G_i; or
   (b) if there exists a node z, $z \in G_i$, and a node u, $u \in G_i$, such that the following holds: $(V_y^{Notify} \subset V_u^{Notify}) \wedge (V_z^{Notify} \subset V_u^{Notify}))$, then put y in G_i; or
   (c) if there exists a node z, $z \in G_i$, and a node u, $u \in W - \bigcup_{j=1}^{i} G_j$, such that the following holds: $(V_y^{Notify} \subset V_u^{Notify}) \wedge (V_z^{Notify} \subset V_u^{Notify}))$, then put both y and u in G_i.
3. Increment i and repeat steps 1 to 3 until $\bigcup_{j=1}^{i} G_j = W$.[14]

Then, we get groups $\{G_i, 1 \leq i \leq l\}$. Moreover, for any node x, $x \in G_i$, and any node y, $y \in G_j$, where $1 \leq i \leq l$, $1 \leq j \leq l$, and $i \neq j$, it holds that $V_x^{Notify} \cap V_y^{Notify} = \emptyset$. Otherwise, if $V_x^{Notify} \cap V_y^{Notify} \neq \emptyset$, suppose $i < j$, then according the step 2 above, y would be included in G_i rather than in G_j. Hence, for any two nodes that are in different groups, their joins are independent. Similarly, it can be checked that for any two nodes in a group, their joins are dependent.

Then, for any suffix ω, if $(G_i)_\omega \neq \emptyset$ and $|V_\omega| < K$, $1 \leq i \leq l$, then by Corollary B.3, $(V \cup W)_\omega = (V \cup G_i)_\omega$.

Consider any node x. If $|V_{j\cdot x[i-1]...x[0]}| \geq K$, then $N_x(i,j).size = K$ since initially $\langle V, \mathcal{N}(V)\rangle$ is K-consistent. If $|V_{j\cdot x[i-1]...x[0]}| < K$ and $W_{j\cdot x[i-1]...x[0]} = \emptyset$, then $N_x(i,j).size = |V_{j\cdot x[i-1]...x[0]}| = |(V \cup W)_{j\cdot x[i-1]...x[0]}|$.

If $|V_{j\cdot x[i-1]...x[0]}| < K$ and $W_{j\cdot x[i-1]...x[0]} \neq \emptyset$, then $|(V \cup W)_{j\cdot x[i-1]...x[0]}| = |(V \cup G_f)_{j\cdot x[i-1]...x[0]}|$, where $(G_f)_{j\cdot x[i-1]...x[0]} \neq \emptyset$. By Lemma 6.4, $N_x(i,j).size = \min(K, |(V \cup G_f)_{j\cdot x[i-1]...x[0]}|)$, hence, $N_x(i,j).size = \min(K, |(V \cup W)_{j\cdot x[i-1]...x[0]}|)$. ■

[14]In [8], we presented an example of how to group nodes following these steps for $K = 1$. See Footnote 16 in [8].

C Proofs of Theorems 4 to 6

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 6.2, 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 4, 5 and 6, and analyses of numbers of messages in sets 4, 5 and 6.[15] Recall that we have defined two functions, $Q_i(r)$ and $P_i(r)$, in Section 6.2.

Lemma C.1 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 1$, join a K-consistent network $\langle V, \mathcal{N}(V)\rangle$. For any node x, $x \in W$, suppose $V_x^{Notify} = V_\omega$. Let $Y = |\omega|$. Then the probability that Y equals i given $|V| = n$, $i \in [b]$, is $P_i(n)$, where $P_i(n)$ is as defined in Definition 6.1.*

Proof: First, let $P_i(n)$ denote the the probability that Y equals i, $i \in [b]$, given $|V| = n$. We next prove that $P_i(n)$ is as defined in Definition 6.1.

If $Y = i$, it indicates that $|V_\omega| \geq K$ and $|V_{x[i]\cdot\omega}| < K$. Thus, $P_i(n) = P(|V_\omega| \geq K \wedge |V_{x[i]\cdot\omega}| < K)$, i.e., $P_i(n) = P(|V_{x[i-1]...x[0]}| \geq K \wedge |V_{x[i]...x[0]}| < K)$.

Next, we compute $P_i(n)$, for $0 \leq i \leq d-1$. In general, IDs of nodes in V are 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 $|V_{x[0]}| < K$, i.e., there is less than K nodes in V with suffix $x[0]$. Suppose there are h nodes in V with suffix $x[0]$, $0 \leq h < K$. Then, IDs of these h nodes are drawn from $b^{d-1} - 1$ possible values (all possible IDs with suffix $x[0]$ except $x.ID$); while IDs of the other $n - h$ nodes are drawn from $b^d - b^{d-1}$ values, $n = |V|$. Therefore,

$$P_0(n) = \frac{\sum_{j=0}^{K-1} C(b^{d-1}-1, j)C(b^d - b^{d-1}, n-j)}{C(b^d - 1, n)}$$

If $1 \leq i < d-1$, then $|V_{x[i-1]...x[0]} \geq K|$ and $|V_{x[i]...x[0]}| < K$. That is, there are only h nodes in V with suffix $x[i]...x[0]$, where $0 \leq h < K$, however, there are H nodes in V with suffix $x[i-1]...x[0]$, $K \leq H \leq n$. Then, IDs of the h nodes with suffix $x[i]...x[0]$ are drawn from $b^{d-i-1} - 1$ possible values (any ID with suffix $x[i]...x[0]$ except $x.ID$), $H - h$ IDs are drawn from $(b-1)b^{d-i-1}$ possible values (any ID that has suffix $x[i-1]...x[0]$ but does not have suffix $x[i]...x[0]$), and $n - H$ IDs are drawn from $b^d - b^{d-i}$ possible values (any ID that does not have suffix $x[i-1]...x[0]$). Let $B = (b-1)b^{d-i-1}$. Hence, for $1 \leq i < d-1$, $P_i(n)$ is

$$\frac{\sum_{j=0}^{K-1} C(b^{d-1-i}-1, j)\sum_{k=K-j}^{\min(n,B)} C(B,k)C(b^d - b^{d-i}, n-k-j)}{C(b^d-1, n)}$$

[15]The messages in sets 5 and 6 can be piggy-backed by probing messages to further save communication costs.

Finally, for $i = d-1$, since each ID is unique, $x.ID$ is different than the ID of any node in V. Therefore, $|V_{x[d-1]...x[0]}| = 0$, and thus $|V_{x[d-1]...x[0]}| < K$ is always true for $K \geq 1$.

$$\begin{aligned} P_{d-1}(n) &= P(|V_{x[d-1]...x[0]}| < K \wedge |V_{x[d-2]...x[0]}| \geq K) \\ &= P(|V_{x[d-2]...x[0]}| \geq K) \\ &= 1 - P(|V_{x[d-2]...x[0]}| < K) \\ &= 1 - P(|V_{x[0]}| < K \\ &\quad \vee (|V_{x[0]}| \geq K \wedge |V_{x[1]x[0]}| < K) \vee ... \\ &\quad \vee (|V_{x[d-3]...x[0]}| \geq K \wedge |V_{x[d-2]...x[0]}| < K)) \\ &= 1 - \sum_{i=0}^{d-2} P_i(n) \end{aligned}$$

Therefore, $P_i(n)$ is as defined in Definition 6.1, $i \in [b]$. ■

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

Proof: In status *copying*, x sends *CP* to nodes g_0, g_1,..., until x receives a *CPRly* from node g_i, such that x finds that there exists an attach-level for itself in $g_i.table$. Note that $g_{i'}$ shares at least one more digit with x than $g_{i'-1}$, for all $1 \leq i' \leq i$. Then, x sends *JW* to g_i, g_{i+1}, ..., until x receives a positive *JWRly* from node g_h. Again, $g_{i'}$ shares at least one more digit with x than $g_{i'-1}$ for all $i+1 \leq i' \leq h$. Hence, the number of *CP* x has sent is $i+1$, and the number of *JW* x has sent is $h-i+1$. The total number of *CP* and *JW* x has sent is $h+2$.

Let $(V \cup W')_x^{Notify} = V_\omega$, where $W' = W - \{x\}$. Assume $|\omega| = j$. Then, in the worst case, x sends *CP* to nodes $\{g_0, g_1, ..., g_i\}$, where $g_{i'}$ shares exactly i' digits with x for all $1 \leq i' \leq i$ (that is, $g_{i'}$ only shares one more digit with x than $g_{i'-1}$). Then, x sends *CP* to nodes $\{g_i, g_{i+1}, ..., g_j\}$, where $g_{i'}$ shares i' digits with x for all $i+1 \leq i' \leq h$. Since $(V \cup W')_x^{Notify} = V_\omega$ and $|\omega| = j$, when x sends a *JW* to g_j, which is a node that shares j digits with it, there must exist an attch-level in the table of g_j for x. According to the above analysis, the total number of *CP* and *JW* x has sent is $j+2$, assuming $|\omega| = j$.

Let $Y = |\omega|$. Similarly to the proof of Lemma C.1, it can be shown that the probability that Y equals j is $P_j(n+m-1)$ ($|V \cup W'| = n+m-1$). Let Z be the total number of *CP* and *JW* x has sent. Therefore, we have

$$\begin{aligned} E(Z) &= E(E(Z|Y)) \\ &= \sum_{i=0}^{d-1}(E(Z|Y=i))P_i(n+m-1) \\ &= \sum_{i=0}^{d-1}(i+2)P_i(n+m-1) \end{aligned}$$

■

Theorem 5 *Suppose node x joins a K-consistent network $\langle V, \mathcal{N}(V)\rangle$, $|V| = n$. Then, the expected number of JoinNotiMsg sent by x is $\sum_{i=0}^{d-1} Q_i(n-K)P_i(n) - 1$.*

Proof: Suppose $V_x^{Notify} = V_\omega$. Then x needs to notify all the nodes in V_ω. By Proposition B.5, there exists a node u_x, $u_x = A(x)$. Then, x sends a *JW* to u_x, however, x sends *JN* to any other node in V_ω (by Proposition B.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 $Z = |V_\omega|$. By Lemma C.1, the probability that Y equals i is $P_i(n)$, given $|V| = n$.

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

We next derive $E(Z|Y=i))$. $Y = i$ indicates that $V_\omega = V_{x[i-1]...x[0]}$. Since $V_x^{Notify} = V_\omega$, we know $|V_\omega| \geq K$, that is, $|V_{x[i-1]...x[0]}| \geq K$. Therefore, among the nodes in V, at least K of them have suffix $x[i-1]...x[0]$ in their IDs. Let X be the expected number of nodes with suffix $x[i-1]...x[0]$ among the remaining $n-K$ nodes in V. Thus, $E(Z|Y=i) = K + E(X)$. Suppose there are j nodes among the $n-K$ nodes that have suffix $x[i-1]...x[0]$. Then j could be any value from 0 to $\min(n-K, b^{d-i}-K-1)$. IDs of these j nodes are drawn from $b^{d-i}-K-1$ possible values (there all b^{d-i} all possible IDs with suffix $x[i-1]...x[0]$, and K of them are already assigned to K nodes in V, and one is assigned to x). IDs of the remaining $n-K-j$ nodes are drawn from $b^d - b^{d-i}$ possible values. Hence, $E(X)$ is

$$\frac{\sum_{j=0}^{\min(n-K, b^{d-i}-K-1)} C(b^{d-i}-K-1, j)C(b^d - b^{d-i}, n-K-j)}{C(b^d-K-1, n-K)}$$

That is, $E(Z|Y=i) = Q_i(n-K)$.[16] Plug $E(Z|Y=i)$ into Equation 1, we get $E(Z)$. The expected number of *JN* x sends during its join is $E(Z)-1$. ■

Theorem 6 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 1$, join a K-consistent network $\langle V, \mathcal{N}(V)\rangle$, $|V| = n$. Then for any node x, $x \in W$, an upper bound of the expected number of JoinNotiMsg sent by x is $\sum_{i=0}^{d-1} Q_i(n+m-1-K)P_i(n)$.*

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 $Y = |\omega|$. By Lemma C.1, the probability that Y equals i, $i \in [d]$, is $P_i(n)$, given $|V| = n$. No matter how many nodes join concurrently with x, $x.att\_level \geq Y$. Moreover, x only sends *JN* to a subset of nodes whose IDs have suffix $x[k-1]...x[0]$, excluding node x itself, where $k = x.att\_level$. These nodes are a subset of nodes with suffix ω. Let $Z = |(V \cup W)_\omega - \{x\}|$. Hence, $J < Z$, which is true for every joining node. Therefore, $E(J) < E(Z)$. To compute $E(Z)$, we have

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

Since $V_x^{Notify} = V_\omega$, we know $|V_\omega| \geq K$, that is, $|V_{x[i-1]...x[0]}| \geq K$. Therefore, among the nodes in V, at least K of them have suffix $x[i-1]...x[0]$ in their IDs. Let X be

[16]If $b^d \gg n-K$, then $E(X) \simeq \frac{(n-K)}{b^i}$. That is, the ID space can be consider as b^i bins, with $x[i-1]...x[0]$ being one of them. Each bin has a capacity limitation of $b^d - b^{d-i}$. Assigning $n-K$ IDs randomly can be considered as throwing $n-K$ balls into the bins randomly. Thus, the expected number of balls falling into bin $x[i-1]...x[0]$ is $\frac{(n-K)}{b^i}$, if none of the bins were overflowed in the process.

the expected number of nodes with suffix $x[i-1]...x[0]$ in the remaining $n-K$ nodes in V, plus the expected number of nodes with suffix $x[i-1]...x[0]$ in $W-\{x\}$. That is, X is the expected number of nodes with suffix $x[i-1]...x[0]$ among $n-K+m-1$ nodes. Similar to the proof of Theorem 5, we have $E(Z|Y=i) = K + E(X) = Q_i(n+m-1-K)$. Plug $E(Z|Y=i)$ to the above equation, we get $E(Z)$, which is an upper bound of $E(J)$, the expected number of *JN* sent by a joining node. ■

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.[17]

Corollary C.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, $x \in W$, an upper bound of the expected number of messages in the form of SN(x, y) or SNRly(x, y) sent during $[t_x^b, t_x^e]$ is $K-1+\sum_{i=0}^{d-1}(\frac{m}{b^{i+1}}+K-1)(d-i-1)P_i(n)$, where $n = |V|$.*

Proof: Consider any node x, $x \in W$. Suppose $V_x^{Notify} = V_\omega$, $|\omega| = i$, and $j = x.att\_level$, then $j \geq i$. Let $D = \{y$, *SN*(x, y) is sent out by x during $[t_x^b, t_x^e]\}$. (Recall that t_x^b is the time x starts joining, and t_x^e is the time x becomes an S-node, as defined in Section 4.1.) Then, for a particular y, $y \in D$, *SN*(x, y) is only sent out by x once. Any y, $y \in D$, must share suffix $x[j]...x[0]$ with x. Thus, $D < |(V \cup W)_{x[j]...x[0]}| \leq |(V \cup W)_{x[i]...x[0]}|$.

Let $Y = |\omega|$. Then $|\omega| = i$ indicates $Y = i$. Let S be the total number of *SN*(x, y) or *SNRly*(x, y) sent during the join process of x, $y \neq x$. Let $Z = |(V \cup W)_{x[i]...x[0]}|$. Then the number of message in the form of *SN*(x, y) initiated by x is at most $Z-1$ (x will not send out *SN*(x, x)). Since $|V_{x[i]...x[0]}| < K$, we know $Z \leq (K-1) + |W_{x[i]...x[0]}|$. For each *SN* sent out by x, it can be forwarded at most $d-i-2$ times (which includes the first time that it is sent out by x). This is because for each y, $y \in D$, that the first receiver of the message shares at least $i+2$ digits with y (both IDs of y and the first receiver must have suffix $y[i+1]...y[0]$), 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 $X = |W_{x[i]...x[0]}|$, the expect number of nodes in W whose IDs have suffix $x[i-1]...x[0]$. We have $X = \frac{m}{b^{i+1}}$. Hence, $E(X|Y=i) = \frac{m}{b^{i+1}}(d-i-2+1)$. Summarize the results, we get

$$\begin{aligned} E(S) &= \sum_{i=0}^{d-1}(E(D|Y=i))P_i(n) \\ &< \sum_{i=0}^{d-1}(E(Z|Y=i))P_i(n) \\ &\leq \sum_{i=0}^{d-1}(E((K-1+X)|Y=i))P_i(n) \\ &= \sum_{i=0}^{d-1}(K-1+\frac{m}{b^{i+1}})(d-i-2)P_i(n) \end{aligned}$$

■

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 $N_y(k', x[k']).size = K$ before x joins, 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 6, this upper-bound is $\sum_{i=0}^{d-1} Q_i(n+m-1-K)P_i(n)$.

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 piggy-backed 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]We observe from simulations that it is rarely the case that a node sends out an *SN*.