Silk: A Resilient Routing Fabric for Peer-to-Peer Networks*

Simon S. Lam and Huaiyu Liu
Dept. of Computer Sciences, Univ. of Texas at Austin, Austin, TX 78712
{lam, huaiyu}@cs.utexas.edu

TR-03-13 May 16, 2003
Revised, October 2003

Abstract

Several proposed peer-to-peer networks use hypercube routing for scalability. In a previous paper, we showed that consistency of neighbor tables in hypercube routing guarantees the existence of a path from any source node to any destination node. Consistency, however, can be broken by the failure of one node. To improve the robustness of hypercube routing, we generalize the concept of consistency *to* K*-*consistency *for* $K \geq 1$*. We then show that a* K*-consistent hypercube routing network provides at least* K *disjoint paths from any source node to any destination node with a probability close to 1. The first objective of this report is the design and specification of a new join protocol together with a proof that it generates* K*-consistent neighbor tables for an arbitrary number of concurrent joins (under the assumption that there is no concurrent leave or failure). To do so, we construct a more general definition of* C-set tree *than our previous one as the conceptual foundation for protocol design and reasoning about* K*-consistency. Both the new protocol and proof require major extensions to the ones in our previous paper to generalize them from 1-consistency to* K*-consistency.*

The second objective of this report is the design and evaluation of a failure recovery protocol for K*-consistent networks. From simulation experiments in which up to 50% of the nodes in a* K*-consistent network failed (when a node fails, it becomes silent), we found that, for* $K \geq 2$*, K-consistency was recovered in every experiment. The third objective of this report is to extend our join and failure recovery protocols such that they construct and maintain* K*-consistent neighbor tables for networks whose nodes join and fail concurrently and frequently. In particular, our join protocol is extended with rules to handle failures of not only existing nodes but also other joining nodes. These extended protocols, being implemented in our prototype system named Silk, will be referred to as Silk protocols. From simulation experiments in which the number of concurrent joins and failures was up to 50% of the initial network size, we found that, for* $K \geq 2$*, Silk generated and maintained K-consistent neighbor tables after the concurrent joins and failures in every experiment. We also present an analysis of the communication and storage overheads of Silk protocols and show that Silk is scalable to a large number of network nodes.*

1 Introduction

Structured peer-to-peer networks are being investigated as a platform for building large-scale distributed systems [8, 9, 12, 11, 14, 5]. These networks use location-independent naming and routing schemes. Specifically, consider a set of (overlay network) nodes and a set of objects. Nodes and objects have identifiers (IDs), which are fixed-length random binary strings.[1] Node and object IDs are drawn from the same ID space. Object IDs are also called *names*. The primary function of each scheme [8, 9, 12, 11, 14, 5] is *name resolution*, that is, mapping a name to a node. (In the literature, names are also referred to as *keys*, and name resolution referred to as *key-based routing*.)

For efficient routing in these schemes, each node maintains $O(\log n)$ pointers to other nodes, to be called neighbor pointers, where n is the number of network nodes. To resolve a name, the average number of application-level hops required is $O(\log n)$ for every scheme except CAN [9], which maintains r pointers per node and routing takes $O(rn^{1/r})$ hops. For these schemes, a simple measure of scalability is the number of application-level hops between any two nodes. However, the average distance traveled for each hop in the underlying Internet (*locality*) is also important. Various ideas have been proposed to improve routing locality [3, 2, 10, 1].

An important problem that has not been addressed adequately is the design and specification of protocols together with a proof that they construct and maintain *consistent* neighbor tables (tables containing neighbor pointers) for network nodes that may join, leave, and fail concurrently. Of interest in this report is a hypercube routing scheme used in several proposed peer-to-peer systems [8, 11, 14, 5]. To implement the

*Research sponsored by NSF grant no. ANI-9977267 and Texas Advanced Research Program grant no. 003658-0439-2001.

[1] These IDs are typically generated using a hash function, such as MD5 or SHA-1. Each ID can be represented as a sequence of d digits of base b.

hypercube routing scheme in a dynamic, distributed environment, we need to address the following problems:[2]

- 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.)
- 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.
- A protocol is needed for nodes to optimize their neighbor tables.

Solving all of these problems is beyond the scope of a single report. In a previous paper [7], we began by defining consistency and constructing a conceptual foundation, called *C-set trees*, for protocol design and reasoning about consistency of neighbor tables in hypercube routing. We also specified a join protocol and constructed a rigorous proof that the join protocol generates consistent neighbor tables for an *arbitrary number of concurrent joins*. The crux of our proof is based upon induction on a C-set tree.

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 small n 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 (or prefix) required by a table entry can be stored in the table entry (see Section 2).

We introduce in this report *K-consistency*, $K \geq 1$, which generalizes *consistency* defined in our previous paper [7] (1-consistency is the same as consistency). Informally, neighbor tables are K-consistent if and only if each table entry stores min(K,H) neighbors, where H is the number of qualified nodes in the network for that table entry (a more precise definition is given in Section 3). It is easy to see that for $H \geq 0$, K-consistent neighbor tables can be constructed for any realization of node IDs. Moreover, in Section 3, we show that a K-consistent network provides at least K disjoint paths from any source node to any destination node with a probability close to 1 provided that n is not too small (e.g., the probability is higher than 0.99 for $n = 300$ and $K = 3$.)

The first objective of this report 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. To do so, we first construct a more general definition of C-set tree than the one in [7]. While the new protocol to be designed is similar in structure to the one in [7], major extensions are needed to generalize it from *1-consistency* to *K-consistency*. In particular, in a 1-consistent network, each neighbor is stored at only one level of the neighbor table of a node.[3] For $K \geq 2$, however, it is possible for a table to store the same neighbor at multiple levels; as a result, we introduced a new concept (the *lowest attach-level* of a node) to ensure protocol correctness. We then construct a rigorous proof that the new protocol generates K-consistent neighbor tables, for $K \geq 1$, after an arbitrary number of concurrent joins (assuming reliable message delivery and no node failure or leave). The structure of the proof, based upon induction on C-set trees, is similar to the one in [7] with major extensions added to generalize it from *1-consistency* to *K-consistency*. We also analyze the expected communication cost of a join.

The second objective of this report is the design and evaluation of a failure recovery protocol, which handles recovery from voluntary leave as a special case, for K-consistent networks.[4] For $K \geq 2$, we found a simple protocol based upon local information that is very effective for failure recovery. From 2,080 simulation experiments in which up to 50% of network nodes failed, we found that all "recoverable holes" in neighbor tables due to failed nodes were recovered by our protocol for $K \geq 2$, that is, the neighbor tables regained K-consistency after the failures in *every* experiment.

The third objective of this report is to extend our join and failure recovery protocols such that they construct and maintain K-consistent neighbor tables for networks whose nodes join and fail concurrently and frequently. In particular our failure recovery protocol is extended to distinguish between "transient" nodes and "stable" nodes (T-nodes and S-nodes, respectively, defined in Section 4.2). Our join protocol is extended with rules to handle failures of both stable nodes and transient nodes. These extended protocols, being implemented in our prototype system named Silk, will be referred to as Silk protocols. We ran 980 simulation experiments in which the number of concurrent joins and failures was up to 50% of the initial network size. We found that, for $K \geq 2$, Silk constructed and maintained K-consistent neighbor tables after the concurrent joins and failures in *every* experiment.

1.1 Related work

In PRR [8], a static set of nodes and preexistence of consistent and optimal neighbor tables are assumed. CAN [9], Pastry [11], and SPRR [5] each has join, leave, and failure recovery protocols, but the issue of neighbor table consistency was not explicitly addressed. In Chord [12], 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

[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 our scheme, the node itself is stored at every level of its own table.

[4] When a node fails, it becomes silent. We do not consider Byzantine failures in this report.

consistency of just one neighbor pointer per node ("successor pointer"), which is sufficient to guarantee correctness of name resolution.

In Tapestry [2], 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 this approach, many existing nodes have to store and process extra states as well as send and receive messages on behalf of joining nodes.

Storing several qualified nodes in each neighbor table entry was first suggested in PRR [8] to facilitate the location of replicated objects. In Tapestry [14], 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, nor protocols specified to construct K-consistent neighbor tables.

1.2 Report organization

The balance of this report is organized as follows. In Section 2, we present an overview of the hypercube routing scheme. In Section 3, we present definitions of consistency and K-consistency, and show that a K-consistent network provides at least K disjoint paths from any source node to any destination node provided that n is not too small. In Section 4, we present a generalized definition of C-set tree, introduce the concept of lowest attach-level, and present a new join protocol. (A pseudocode specification of the protocol is in Appendix A.) The correctness properties and an analysis of the communication cost of the new join protocol are presented in Theorems 2 to 6. In Section 5, we present our failure recovery protocol together with extensive simulation results. In Section 6, we present protocol extensions to handle concurrent joins and failures (namely, Silk protocols) together with extensive simulation results. In Section 7, we further investigate storage and communication overheads of Silk as a function of K. We conclude in Section 8.

2 Overview of Hypercube Routing Scheme

In this section, we briefly introduce the hypercube routing scheme, following the notation and terminology in PRR [8]. 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. For example, a 160-bit ID can be represented by 40 Hex digits ($d = 40$, $b = 16$). Hereafter, we use $x.ID$ to denote the ID of node x.

Given a message with destination node ID, $z.ID$, the objective of each hypercube routing step is to forward the message from its current node, say x, to a next node, say y, such that the suffix (or prefix) 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, name resolution is achieved in $O(\log_b n)$ steps on the average and d steps in the worst case.

To implement hypercube routing, each node maintains a data structure, called *neighbor table*. The entry j at level i, $0 \leq j \leq b-1$, $0 \leq i \leq d-1$, referred to as the (i, j)-entry, in the table of node x contains link information to nodes whose IDs and $x.ID$ share a common suffix of length i digits, and whose ith digit is j.[6] These nodes are said to be *neighbors* of x. The link information for each neighbor consists of the neighbor's ID and its IP address. For simplicity, we will use "neighbor" or "node" instead of "node's ID and IP address" whenever the meaning is clear from context.

If multiple nodes exist with the required suffix of the (i, j)-entry, then a subset of these nodes, chosen according to some criterion, may be stored in the entry with the "nearest" one designated as the *primary(i, j)-neighbor*. Each node also keeps track of its *reverse-neighbors*. Node x is a reverse(i, j)-neighbor of node y if y is a (i, j)-neighbor of x. Figure 1 shows an example neighbor table, where IP addresses of the neighbors are omitted. The number 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 required suffix. Consider a message being routed from source 21233 to a destination node, say 03231. The message is first forwarded to the primary$(0, 1)$-neighbor of 21233, which is 33121 in Figure 1, then to the primary$(1, 3)$-neighbor of 33121, say, 13331, and so on, until it reaches 03231.

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

∧	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
level 4		level 3		level 2		level 1		level 0	

Figure 1. An example neighbor table

3 K-consistent Networks

Constructing and maintaining consistent neighbor tables is an important design objective for structured peer-to-peer overlay networks. We next present a rigorous definition of consistency from [7] and then introduce a stronger property, K-consistency, for the hypercube routing scheme. Table 1 presents notation used throughout this report.

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

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

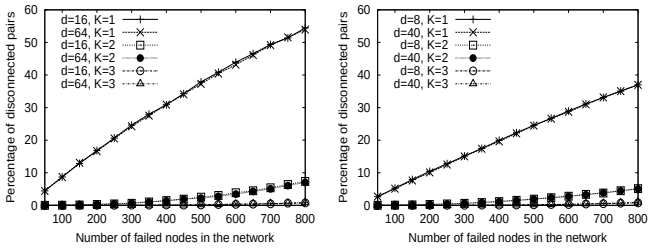
Intuitively, part (a) in the above definition states that for each table entry, if there exists at least one node in the network

[5] In this report, we follow PRR [8] and use suffix matching, whereas Pastry [11], Tapestry [14], and SPRR [5] use prefix matching. The choice is arbitrary and conceptually insignificant.

[6] We count digits in an ID from right to left, with the 0th digit being the *rightmost* digit.

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
$\|\omega\|$	the number of digits in 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
$V_{l_i...l_0}$	a *suffix set* of V, which is the set of nodes in V, each of which has an ID with the suffix $l_i...l_0$
$\|V\|$	the number of nodes in set V

Table 1. Notation



(a) b=4, n=4000 (b) b=16, n=4000

Figure 2. Percentage of disconnected source-destination pairs for different K values

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.*

In a consistent network, every node is reachable from every other node. If many nodes may fail in a network, an approach to improve robustness is to store in each table entry multiple *qualified nodes*, i.e., nodes with the required suffix for the entry. 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 $N_x(i,j).size = \min(K, |V_{j \cdot x[i-1]...x[0]}|)$, $i \in [d]$, $j \in [b]$, and $N_x(i,j) \subseteq V_{j \cdot x[i-1]...x[0]}$.

(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 is a generalization of Definition 3.1. Intuitively, part (a) states that if set $V_{j \cdot x[i-1]...x[0]}$ (the set of nodes in V with suffix $j \cdot x[i-1]...x[0]$) is not empty, then $N_x(i,j)$ should be filled with either K nodes in $V_{j \cdot x[i-1]...x[0]}$ or all of the nodes if $|V_{j \cdot x[i-1]...x[0]}| < K$. Part (b) remains the same. 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 report, for each node x, we choose $N_x(i, x[i]).first$ to be x itself, $i \in [d]$, for efficient routing.

To study the robustness of K-consistent networks in the presence of failures, we conducted simulation experiments as follows: For every node in a network of n nodes, each entry in its neighbor table was filled with K neighbors if there were K or more qualified nodes in the network for that entry; otherwise, all qualified nodes (if any) were stored in the entry. We 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, where f is the number of failed nodes and the maximum value of f is 20% of n. For each combination, we ran five simulations and calculated the average value of the percentage of source-destination pairs that became disconnected. Figure 2 shows some simulation results. First, note that the results are insensitive to the value of d. In each plot, for each K value, the two curves for two different d values are almost the same. In Figure 2, we can distinguish three curves in each plot, with the top curve being $K = 1$ and the bottom $K = 3$. 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 robust 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 Section 5.)

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, we say that two paths from source node x to destination node y are *disjoint* iff 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** iff every pair of paths in the set are disjoint. For example, let a, b, and c denote nodes. Then the following paths are disjoint: $x \rightarrow y$, $x \rightarrow a \rightarrow y$, and $x \rightarrow b \rightarrow c \rightarrow y$. [7]

[7] Note that nodes here are user machines in a peer-to-peer network. Thus, it is possible for two disjoint paths in a hypercube routing network to share a

Theorem 1 *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 $x \neq y$, 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 B.1.

Lemma 3.1 *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 3.1 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 3.2 summarizes all the cases.

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 $y \notin N_x(0, x[0])$, 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 $y \in x.table \wedge y \notin N_x(0, x[0])$). For any event X, let $P(X)$ denote the probability of X. We first derive $P(A \wedge B)$.

$$P(A \wedge B) = P(A|B)P(B)$$

$P(A|B)$ 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 3.2, 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 $y[0]$.

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

We next derive $P(B)$. Let K' be the number of neighbors stored in $N_x(0, x[0])$ other than x itself. Then $K' \leq K - 1$.

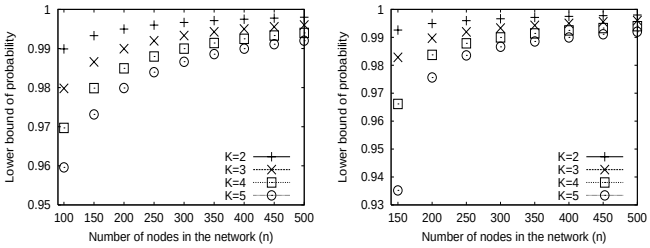
$$P(B) = 1 - P(y \in N_x(0, x[0])) \geq 1 - \frac{K-1}{n-1}$$

Combining the above results, we have

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

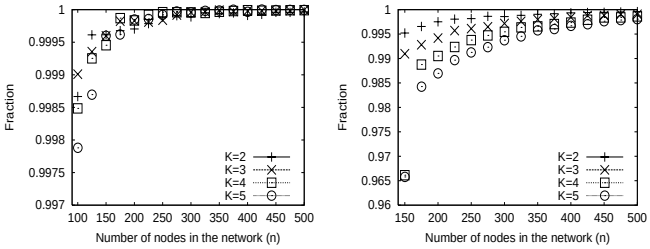
■

Figure 3 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.[8] Observe that when n increases, the probability lower bound approaches 1. For example, the lower bound is higher than 0.99 for $n = 300$ and $K = 3$.



(a) b = 4, d = 64 (b) b = 16, d = 40

Figure 3. Lower bound of the probability that there exist at least K disjoint paths for every source-destination pair



(a) b = 4, d = 64 (b) b = 16, d = 40

Figure 4. Simulation results on the fraction of source-destination pairs with at least K disjoint paths

To validate the above analysis, we conducted simulations to evaluate the number of disjoint paths for each source-destination pair in a K-consistent network. In each simulation, there was a set V of n nodes, each with a randomly generated ID. The neighbor table of each node was constructed according to Definition 3.3 and $N_x(i, x[i]).first = x$ 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 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 presents our simulation results. Observe that the results in Figure 4 are much closer to 1 than the corresponding lower bound results in Figure 3, as expected. For example, the fraction of source-destination pairs with at least K disjoint paths is greater than 0.996 for $n = 300$, $K = 3$, and $b = 16$ in Figure 4(b) and greater than 0.9999 for $n = 300$, $K = 3$, and $b = 4$ in Figure 4(a).

4 Join Protocol for K-consistency

Analysis and simulation results in the previous section demonstrate the advantages of K-consistency. We next design a new join protocol that constructs and maintains K-consistent neighbor tables for concurrent joins. Design of the join protocol for K-consistency, $K \geq 1$, is based on our prior work for

router in the underlying Internet. This would not be a reliability concern since routers are generally much more resilient than user machines.

[8] We observed that this probability is insensitive to the value of d. Differences between results in (a) and (b) of Figure 3 for the same K are due to different b values for (a) and (b).

$K = 1$ [7]. Major extensions are needed, which are presented in Sections 4.1 and 4.2.

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.

4.1 Generalized C-set tree

In [7], we defined *C-set trees* as a conceptual foundation for reasoning about 1-consistency and guiding our protocol design. In this section, we present generalized definitions as well as correctness conditions for reasoning about K-consistency. We begin with a generalized definition for the *notification set of x regarding V*, denoted by V_x^{Notify}. Suppose node x joins a network $\langle V, \mathcal{N}(V)\rangle$. Then, intuitively, V_x^{Notify} is the set of nodes in V that need to update their tables if x were the only node that joins $\langle V, \mathcal{N}(V)\rangle$.

Definition 4.1 *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, $x \in W$, 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.*

Given V and x, as K is increased from 1, the set V_x^{Notify} may get larger (never smaller). For instance, suppose x =10261 ($b = 8, d = 5$), and $V = \{13061, 31701, 00261, 10353\}$. If $K = 1$, then $V_x^{Notify} = \{00261\}$; if $K = 2$, then $V_x^{Notify} = \{00261, 13061\}$; if $K = 3$, then $V_x^{Notify} = \{00261, 13061, 31701\}$.

Next, we introduce the generalized concept of a C-set tree. When a set of nodes W join a K-consistent network $\langle V, \mathcal{N}(V)\rangle$, the tasks of a join protocol are to update neighbor tables of nodes in V and to construct tables for nodes in W. A joining node can copy neighbor information from nodes in V to reach nodes in V. However, how to establish neighbor pointers from nodes in V to nodes in W and between nodes in W is a more complex task. We use C-set trees as a conceptual tool that guides protocol design to establish these pointers. Intuitively, a C-set tree organizes nodes in V that need to update their tables and nodes in W into a tree, if the notification sets regarding V (*noti-sets*, in short) of all joining nodes are the same. Generally, the noti-sets of all nodes in W may not be the same. Then, nodes in W with the same noti-set belong to the same C-set tree and the C-set trees for all nodes in W form a forest. Each C-set tree in the forest can be treated separately in proving protocol correctness. In the balance of this subsection, our discussion is focused on a single C-set tree. We next present the generalized definitions of a *C-set tree template* and a *C-set tree realization*.

Definition 4.2 *Suppose a set of nodes, $W = \{x_1, ..., x_m\}$, $m \geq 1$, join a K-consistent network $\langle V, \mathcal{N}(V)\rangle$, and for any node x, $x \in W$, $V_x^{Notify} = V_\omega$, where $|\omega| = k$. Then the C-set* **tree template** *associated with V and W, denoted by $C(V, W, K)$, 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 $C_{l_{j-1} ... l_1 \cdot \omega}$.*

Given V, W and K, the tree template is determined. For example, suppose a set of nodes ($b = 8, d = 5$), 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 $C(V, W, K)$ is as shown in Figure 5(a). The value of K affects the tree template through the noti-sets of nodes in W. Suppose $K = 1$ in the above example, then nodes {41633, 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.

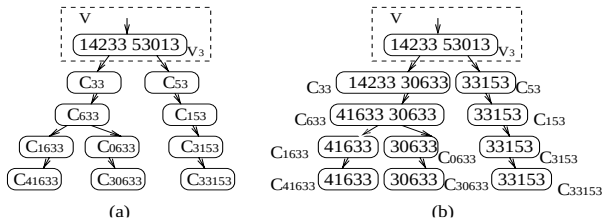


Figure 5. C-set tree

The task of the join protocol is to construct and update neighbor tables such that paths are established between nodes; *conceptually* nodes are filled into each C-set in $C(V, W, K)$. For example, when 14233 updates its (1,3)-entry and fills 30633 into the entry, then conceptually 30633 is filled into C_{33}. We use $cset(V, W, K)$ to denote the C-set tree realized at the end of all joins, defined as follows. Hereafter, let t_x^e denote the end of the joining period [7] of x, and t^e denote $\max(t_{x_1}^e, ..., t_{x_m}^e)$.

Definition 4.3 *Suppose a set of nodes, $W = \{x_1, ..., x_m\}$, $m \geq 2$, join a K-consistent network $\langle V, \mathcal{N}(V)\rangle$, 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, is defined as follows:*

- *V_ω is the root of the tree.*
- *$C_{l_1 \cdot \omega}$ is a child of V_ω, $l_1 \in [b]$, if $W_{l_1 \cdot \omega} \neq \emptyset$ and $C_{l_1 \cdot \omega} = \{x, x \in (V \cup W)_{l_1 \cdot \omega} \wedge (\exists u, u \in V_\omega \wedge x \in N_u(k, l_1))\}$ is not empty.*
- *$C_{l_j ... l_1 \cdot \omega}$ is a child of $C_{l_{j-1} ... l_1 \cdot \omega}$, $2 \leq j \leq d - k$, l_1,...,$l_j \in [b]$, if $W_{l_j ... l_1 \cdot \omega} \neq \emptyset$ and $C_{l_j ... l_1 \cdot \omega} = \{x, x \in (V \cup W)_{l_j ... l_1 \cdot \omega} \wedge (\exists u, u \in C_{l_{j-1} ... l_1 \cdot \omega} \wedge x \in N_u(k + j - 1, l_j))\}$ is not empty.*

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). By the end of the joins, neighbor tables of nodes in $V \cup W$ are K-consistent if the following **correctness conditions** hold:

(1) $cset(V, W, K)$ has the same structure as $C(V, W, K)$. Also, for any C-set, $C_{\omega'}$, 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), for each child C-set of V_ω, denoted as $C_{l\cdot\omega}$, $l \in [b]$, y has stored $\min(K, |C_{l\cdot\omega}|)$ nodes with suffix $l\cdot\omega$ in $N_y(k,l)$, $k = |\omega|$.

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

4.2 Lowest attach-level

The join protocol for K-consistency is similar in structure to the one for 1-consistency [7]. The status of a joining node begins in *copying*, then changes to *waiting*, *notifying*, and *in_system* in that order. A node in 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 other nodes to fill in most entries 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 (conceptually) that it finds a position for itself 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.

However, there are major differences between the new join protocol and the one in [7]. The first and the most obvious difference is that in the join protocol for K-consistency, a joining node, x, needs to search for K neighbors for each of its table entry. Second, the conditions for x to change status from *copying* to *waiting*, and from *waiting* to *notifying* are different, as explained below. Third, while in status *notifying*, the set of nodes x seeks and sends notifications to may become larger for a larger K value (see Section 4.3).

To illustrate the last two differences, we first discuss how a joining node is attached to a K-consistent network. First, for $K = 1$, a neighbor, x, is only stored at one level in the table of a node y (however, y itself is stored at every level of its own table). 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 example, node 00261 is only stored at level-4 in the table of 10261, since 10261 itself is already filled into entries at lower levels (for each suffix of 0261). 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 needed. We call it the *lowest attach-level* of x, or simply the *attach-level* of x for notational convenience.

Definition 4.4 *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 $0 < i \le k$, $N_y(i', x[i']).size < K$ for all i', $i \le i' \le k$, and $N_y(i-1, x[i-1]).size = K$;*
- *an attach-level does not exist if $N_y(k, x[k]).size = K$.*

The attach-level defined above is the *lowest* level at which node x can be stored in $y.table$. One of the conditions for a joining node, x, to change status from *copying* to *waiting* is that when it receives a reply from node y, x finds that there exists an attach-level for itself in the copy of $y.table$ received. The condition for x to change status from *waiting* to *notifying* is that when a node, z, which receives a request from x to store x in $z.table$, finds that there exists an attach-level, say level-j, for x in $z.table$, stores x and sends x a positive reply. Level-j is then called the *attach-level of x in the network*. In status *notifying*, x seeks and notifies nodes that share the rightmost j digits with it. Any node that receives such a notification from x cannot store x into a level that is lower than j. Conceptually, this means that once x is filled into a C-set, it will not be filled into any ancestor of that C-set in its C-set tree.

4.3 Protocol description

Figure 6 presents the state variables of a joining node. Note that each node stores, for each neighbor in its table, 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.

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

The protocol messages are listed in Figure 7. They are similar to those in [7], with the following major extension: An integer is included in *JoinWaitRlyMsg* and *JoinNotiMsg* to explicitly indicate the attach-level of a joining node in the network. In a *JoinWaitRlyMsg*, the integer indicates the attach-level of the receiver, and in a *JoinNotiMsg*, the integer indicates the attach-level of the sender.

Next, we describe the new join protocol informally. A specification of the protocol in pseudocode is given in Appendix A. In status *copying*, a joining node, x, fills in most entries of its table, level by level, as follows. To construct its table at level-i, $i \in [d]$, x needs to find a node, g_i, that shares the rightmost i digits with it and send a *CpRstMsg* to g_i 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. x begins with g_0. From $g_0.table$, x finds a node g_1 that shares the rightmost digit with it, and requests $g_1.table$ from g_1 if such a node exists and is an S-node. From $g_1.table$, x tries to find g_2, and so on.

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

In status *copying*, each time after receiving a *CpRlyMsg* containing a neighbor table from g_i, $i \in [d]$, 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 and node u is a T-node, where $u = N_y(k, x[k]).first$ and $k = |csuf(x.ID, y.ID)|$. 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.

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. When a node, y, receives a *JoinWaitMsg* from x, there are two cases. If y is not an S-node, it stores the message to be processed after it has become an S-node. 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, $k = |csuf(x.ID, y.ID)|$, and sends a *JoinWaitRlyMsg*(*positive*, j, $y.table$) to x, to inform x that the lowest level x is stored is level-j. Level-j is then 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 a negative *JoinWaitRlyMsg* including $y.table$ to x. After receiving the reply (positive or negative), x searches the neighbor table included in the reply for new neighbors to update its own table.

Note that if an attach-level does not exist for x in $y.table$, then even if there is some entry, for which x has the required suffix, is not full (fewer than K neighbors), y will not store x. For example, when node 30061 receives a *JoinWaitMsg* from node 00261, if in the table of node 30061, $(2, 2)$-entry is full (thus an attach-level does not exists for 00261), then even if $(1, 6)$-entry is not full, 30061 will not store 00261 into $(1, 6)$-entry. As shown in our proofs, the $(1, 6)$-entry in this example will eventually be filled up by other nodes.

Upon receiving a negative reply from y, x has to send another *JoinWaitMsg*, this time to u, $u = N_y(k, x[k]).first$. 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*. Note that before x is attached to the network, communication between the network and node x is one-way: x can reach nodes in the network. After x is attached to the network, communication becomes two-way: other nodes already in the network can reach x now.

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 j 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 $j \leq i \leq |csuf(x.ID, z.ID)|$, searches $x.table$ for new neighbors to update z's table, and then replies to x with $z.table$. From the reply, x may find more nodes that share the rightmost j digits with it and send *JoinNotiMsg* to these nodes. Meanwhile, x searches the copy of $z.table$ for new neighbors to update its own table.

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.

4.4 Protocol Analysis

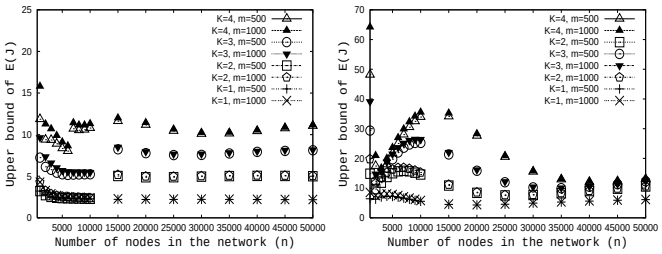
In this section, we present theorems about correctness of the join protocol specified in Section 4.3, and our analytical results for the communication cost of each join. Theorem 2 is proved by induction on generalized C-set trees. Proofs of the theorems are presented in Appendix B.

Theorem 2 states that if a set of nodes use the join protocol to join a K-consistent network, then at the end of the joins, the resulting network is a K-consistent network. Theorem 3 states that the join process of each node eventually terminates. Recall that t^e denotes $\max(t^e_{x_1}, ..., t^e_{x_m})$, where t^e_x denotes the end of the joining period of node x.

Theorem 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, at time t^e, $\langle V \cup W, \mathcal{N}(V \cup W)\rangle$ is a K-consistent network.*

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, each node x, $x \in W$, eventually becomes an S-node.*

Next, we analyze the communication cost of each join. Here we only present results for the number of messages of



(a) b=4, d=64 (b) b=16, d=40

Figure 8. Upper bound of expected number of JoinNotiMsg sent by a joining node versus n for different values of K and m

type *CpRstMsg*, *JoinWaitMsg*, and *JoinNotiMsg*,[9] since these messages may include a copy of a neighbor table and thus could be big in size. Analysis of other types of messages is presented in Appendix B. In Theorem 5, $C(X, Y)$ denotes the number of Y-combinations of X objects.

Theorem 4 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 1$, join a K-consistent network $\langle V, \mathcal{N}(V)\rangle$. Then, for any x, $x \in W$, the total number of CpRstMsg and JoinWaitMsg sent by x is at most $d + 1$.*

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 $\sum_{i=0}^{d-1} \frac{n}{b^i} P_i(n) - 1$, where $P_0(n)$ is $\sum_{j=0}^{K-1} \frac{C(b^{d-1}-1,j)C(b^d-b^{d-1},n-j)}{C(b^d-1,n)}$; $P_i(n)$, for $1 \leq i < d-1$, is $\sum_{j=0}^{K-1} C(b^{d-1-i}-1, j) \sum_{k=K-j}^{\min(n-j,B)} \frac{C(B,k)C(b^d-b^{d-i},n-k-j)}{C(b^d-1,n)}$, where $B = (b-1)b^{d-i-1}$; and $P_{d-1}(n)$ is $1 - \sum_{j=0}^{d-2} P_j(n)$.*

Theorem 6 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, 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 $\sum_{i=0}^{d-1} (\frac{n+m}{b^i}) P_i(n)$, where $P_i(n)$ is defined in Theorem 5.*

Figure 8 plots the upper bound of expected number of *JoinNotiMsg* sent by a joining node according to Theorem 6, where n is the number of nodes in the initial network, m is the number of nodes that join the network, and $E(J)$ is the expected number of *JoinNotiMsg*. Notice that the upper bound increases with K, however, for a fixed value of K, the upper bound is insensitive to the value of m, and stays flat as n becomes large.

4.5 Network initialization

The join protocol can also be used for 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 concurrently. Each node is given x to start with and executes the join protocol. Eventually, a K-consistent network is constructed.

[9] The number of replies to these messages are the same since request and reply are one-to-one related.

5 Failure Recovery

In this section, we design a protocol for nodes to recover from failures of other nodes in a K-consistent network. We consider the "fail-stop" model only, i.e., when a node fails, it becomes silent and stays silent. If some neighbor in a node's table has failed, we assume that the node will detect the failure after a timeout duration, e.g., timeout after sending a periodic probe. Note that the failure of a reverse-neighbor affects neither K-consistency nor consistency of neighbor tables. Therefore, if a reverse-neighbor has failed, the reverse-neighbor pointer is simply deleted without any recovery action. Hence, the protocol being designed is for recovery from neighbor failures only.

Consider a network of n nodes that satisfies K-consistency initially. Suppose f out of the n nodes (chosen randomly) fail at the same time or within a relatively short time duration. Our objective in this section is to design a recovery protocol such that some time after the f failures have occurred, neighbor tables in the remaining $n - f$ nodes satisfy K-consistency again. (In the next section, our protocols will be extended to handle concurrent joins and failures.)

Suppose a node in the network, say y, has failed and y has been stored in the (i, j)-entry of the table of node x. We say that the failure of y leaves a *hole* in the (i, j)-entry of $x.table$. To maintain K-consistency, x needs to find a qualified substitute for y, i.e., x needs to find a node u such that u has the required suffix of the (i, j)-entry in $x.table$, u has not failed, and $u \notin N_x(i, j)$. (It is possible that u fails later and x needs to find a qualified substitute for u.) To determine whether or not the network of $n - f$ remaining nodes satisfies K-consistency, we distinguish between *recoverable holes* and *irrecoverable holes*. A hole in the (i, j)-entry of $x.table$ is **irrecoverable** after the f failures if a qualified substitute does not exist among the $n - f$ remaining nodes, i.e., every node in the set of $n - f$ nodes that has the required suffix of the (i, j)-entry in $x.table$ is already in $N_x(i, j)$.

The *objective of a failure recovery protocol* is to find a qualified substitute for every recoverable hole in each node's neighbor table. Irrecoverable holes, on the other hand, cannot possibly be filled and do not have to be filled, according to Definition 3.3, for neighbor tables to satisfy K-consistency.

A difficulty in failure recovery is that individual nodes do not have global information and cannot distinguish recoverable from irrecoverable holes.[10] We design our recovery process for each hole in a node's table as a sequence of search steps executed by the node based on local information (its neighbors and reverse-neighbors). After the entire sequence of steps has been executed and no qualified substitute is found, the node considers the hole to be irrecoverable and the recovery process terminates. We then evaluate our failure recovery protocol in a large number of simulation experiments. In a simulation experiment, we can check how fast our failure recovery protocol finds a qualified substitute for a recoverable hole. Further-

[10] If the network is not partitioned, a broadcast protocol can be used to search all nodes to determine if a hole is recoverable. A broadcast protocol, of course, is not a scalable approach.

more, we can check how often our failure recovery protocol terminates correctly when it considers a hole to be irrecoverable (since we have global information in simulation).

5.1 Protocol design

Suppose a node, x, detects that a neighbor, y, has failed and left a hole in the (i, j)-entry, $i \in [d]$, $j \in [b]$, in $x.table$. Let ω denote the required suffix of the (i, j)-entry in $x.table$. To find a qualified substitute for y with reasonable cost, we propose a sequence of four steps, (a)-(d) below, based on x's local information. At the beginning of each step, except step (a), x sets a timer. If the timer expires and no qualified substitute for y has been found, then x proceeds to the next step.

To determine whether some node u is a qualified substitute for y, x needs to know whether u has failed. In our protocol, x makes this decision based upon local information. More specifically, x maintains a list of failed nodes it has detected so far.[11] x accepts u as a qualified substitute for y if u is not on the list, u has the required suffix ω, and $u \notin N_x(i, j)$.

Step (a) x deletes y from its table, then searches its neighbors and reverse-neighbors to find a qualified substitute for y.

Step (b) x queries each of the remaining neighbors in the (i, j)-entry of its table (if any). In each query, x includes a copy of nodes in $N_x(i, j)$. When a node, say z, receives such a query from x, it searches its neighbors and reverse-neighbors to find a node that has suffix ω and is not in $N_x(i, j)$. If one is found, z replies to x with the node's ID (and IP address).

Step (c) x queries each of its neighbors at level-i (all entries) including neighbors in the (i, j)-entry, using a protocol same as the one in step (b).

Step (d) x queries every one of its neighbors (all levels) including neighbors at level-i, using a protocol same as the one in step (b).

When the timer in step (d) expires and no qualified substitute has been found, x terminates the recovery process and considers the hole left by y to be irrecoverable. (For clarity, we have described just our basic recovery protocol, without various optimizations that are possible in a protocol implementation.) The earlier a hole is recovered, the less is the communication overhead incurred. If a hole is recovered in step (a), there is no communication overhead. If a hole is recovered in step (b), at most $2(K - 1)$ messages are exchanged, $K - 1$ queries and $K - 1$ replies. If a hole is recovered in step (c), there are at most $2Kb$ messages, plus the messages exchanged in step (b). If a hole is recovered in step (d), approximately $2Kb \log_b n$ messages, plus the messages in steps (b) and (c), are exchanged.

[11] In implementation, a failed node only needs to stay in the list long enough for all its reverse-neighbors to detect its failure. To keep the list from growing without bound, x can delete nodes that have been in the list for a sufficiently long time.

5.2 Simulation results

To evaluate the performance and accuracy of our failure recovery protocol, we conducted 2,080 simulation experiments.[12] We used the GT_ITM package [13] to generate network topologies. For a generated topology with a set of routers, n overlay nodes (end hosts) were attached randomly to the routers. For the simulations reported in Table 2, three topologies were used. The 1000-node and 2000-node simulations used a topology with 1056 routers. The 4000-node simulations used a topology with 2112 routers. The 8000-node simulations used a topology with 8320 routers. 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.[13]

In each simulation, a network of n nodes with K-consistent neighbor tables was first constructed. Then a number, f, of randomly chosen nodes failed. For 1000-node and 8000-node simulations, the f nodes failed at the same time. For 2000-node simulations and each specific K value, the f nodes failed at the same time for 84 out of the 180 experiments; a Poisson process was used to generate failures in the balance of the experiments, with half of the experiments at the rate of 1 failure per second and the other half at the rate of 1 failure every 10 seconds. For comparison, the timeout value used to determine whether a neighbor has failed was 5 seconds, and the timeout value used in each of the protocol steps (b)-(d) was 20 seconds. Therefore, most failure recovery processes ran concurrently even when the Poisson rate was slowed to one failure every ten seconds. For 4000-node experiments and each specific K value, the f nodes failed at the same time in 104 out of the 116 experiments, with a Poisson process at the rate of 1 failure per second used in the balance of the experiments.

We conducted simulations for different combinations of b, d, K, n and f values. For each network of n nodes, $n \in \{1000, 2000, 4000, 8000\}$, four pairs of (b, d) were used, namely: (4,16), (4,64), (16,8), and (16,40).[14] Then, for each (b, d) pair, K was varied from 1 to 5. For each (n, b, d, K) combination, f was varied from $0.05n$ to $0.1n$, $0.15n$, $0.2n$, $0.3n$, $0.4n$, and $0.5n$ (1540 experiments were run for $f = 0.05n$ to $f = 0.2n$, with approximately the same number of experiments for each; 540 experiments were run for $f = 0.3n$ to $f = 0.5n$, with 180 experiments for each). In constructing the initial K-consistent networks for simulations, we experimented with four approaches to choose neighbors for each entry: (i) choose K neighbors randomly from qualified nodes, (ii) choose K closest neighbors from qualified nodes, (iii) choose K neighbors randomly from qualified nodes that are within a

[12] These 2,080 experiments together with the 980 experiments to be presented in Section 6 required several months of execution time on several workstations. A typical experiment took several hours to run on a Linux workstation with 2.66 GHz CPU and 2 GB memory. Each simulation experiment for 8,000 nodes, $b = 16$, and $K \geq 3$ shown in Table 2 took 40 - 72 hours to run.

[13] The maximum end-to-end delay in 8000-node simulations was 969 ms.

[14] In Tapestry, $b = 16$ and $d = 40$, while in Pastry, $b = 16$ and $d = 32$.

multiple of the closest neighbor's distance, (iv) use our join protocol in Section 4 to initialize table entries.

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

Table 2. Results from 2,080 simulation experiments (f was $0.05n$, $0.1n$, $0.15n$, $0.2n$, $0.3n$, $0.4n$ or $0.5n$)

Table 2 shows results of 2,080 simulation experiments. In a simulation, if all recoverable holes are recovered (thus K-consistency maintained) at the end of the simulation, it is recorded as a *perfect recovery* in Table 2. In these simulation experiments, every simulation for $K \geq 2$ finished as a perfect recovery, i.e., every recoverable hole was recovered. Thus in K-consistent networks, for $K \geq 2$, our failure recovery protocol is extremely effective.

Table 3 presents results on the cumulative fraction of recoverable holes that were recovered by the end of each step in the recovery protocol. The results are from ten simulations for a network with 4,000 nodes and 800 failures; the initial neighbor tables were constructed using approach (iii), described above. From Table 3, observe that step (d) in our recovery protocol was seldom used. There was a dramatic improvement in the recovery protocol's performance when K was increased from 1 to 2. Also observe that the fraction of recoverable holes that were recovered after each step increases with K. For $K \geq 2$, more than 93% of recoverable holes were recovered within the first two steps and more than 99.8% within the first three steps. For $K \geq 3$, more than 98.9% of recoverable holes were recovered within the first two steps.

b, d, K	n, f	step (a)	step (b)	step (c)	step (d)
4, 64, 1	4000, 800	0.451594	0.451594	0.920969	0.9988833
4, 64, 2	4000, 800	0.668176	0.938131	0.998077	1.000000
4, 64, 3	4000, 800	0.760213	0.98974	0.998774	1.000000
4, 64, 4	4000, 800	0.816133	0.997837	0.999252	1.000000
4, 64, 5	4000, 800	0.851577	0.999126	0.999736	1.000000
16, 40, 1	4000, 800	0.453649	0.453649	0.999093	1.000000
16, 40, 2	4000, 800	0.633784	0.932868	0.9998539	1.000000
16, 40, 3	4000, 800	0.716517	0.989295	0.9999861	1.000000
16, 40, 4	4000, 800	0.77311	0.997785	1.000000	1.000000
16, 40, 5	4000, 800	0.823924	0.999441	1.000000	1.000000

Table 3. Cumulative fraction of recoverable holes recovered at each step

Table 4 shows the total number of holes, the number of irrecoverable holes, as well as the number of recoverable holes recovered at each step for the same simulation experiments shown in Table 3. Observe from Table 4 that even though the total number of holes increased when K was increased, the number of recoverable holes recovered at step (a) also increased with K. The number of recoverable holes recovered in step (b) did not increase much with K; it actually declined in steps (c) and (d).

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

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

5.3 Voluntary leaves

A voluntary leave can be handled as a special case of node failure if necessary. When a node, say x, leaves, it can actively inform its reverse-neighbors and neighbors. To each reverse-neighbor, x suggests a possible substitute for itself. When a node receives a leave notification from x, for each hole left by x, it checks whether the substitute provided by x is a qualified substitute. If so, the hole is filled with the substitute; otherwise, failure recovery is initiated for the hole left by x.

6 Silk Protocols for Concurrent Joins and Failures

Consider a K-consistent network, $\langle V, \mathcal{N}(V)\rangle$. Suppose a set of new nodes, W, join the network while a set of nodes, F, fail, $F \subset V \cup W$ and $V - F \neq \emptyset$. Our goal in this section is to design extended join and failure recovery protocols such that eventually the join process of each node in $W - F$ terminates and $\langle (V \cup W) - F, \mathcal{N}((V \cup W) - F)\rangle$ is a K-consistent network. These extended protocols will be referred to as Silk protocols. In general, designing a failure recovery protocol to provide perfect recovery is an impossible task; for example, consider a scenario in which an arbitrary number of nodes in $V \cup W$ fail. On the other hand, we observed in Section 5 that the basic failure recovery protocol achieved perfect recovery for K-consistent networks, for $K \geq 2$, in which up to 50% of the nodes failed. This level of performance, we believe, would be adequate for many applications.

Design of Silk's join and failure protocols in this section follows the Lam-Shankar approach [4] on how to compose modules. The service provided by a composition of the two protocols herein is construction and maintenance of K-consistent neighbor tables. The join protocol is designed with the assumption that the failure recovery protocol provides a "perfect recovery" service, that is, for every hole found in the neighbor table of a node, the node calls failure recovery and within a bounded duration, failure recovery returns with a qualified substitute for the hole or the conclusion that the hole is irrecoverable at that time. Following the protocol composition approach in [4], we ensure that progress of the failure recovery protocol

does not depend upon progress of the join protocol. Thus in the extensions to be presented, failure recovery actions are always executed before join actions.

6.1 Protocol extensions

When there are nodes joining a network, the network consists of both S-nodes and T-nodes. Recall that if a node is in status *in_system*, it is an S-node; otherwise, it is a T-node. Silk's extensions to the basic join protocol in Section 4.3 and failure recovery protocol in Section 5.1 are stated as a set of rules.

Rule 1 In filling a table entry with a qualified node, do not choose a T-node unless there is no qualified S-node.

Rule 1 extends the basic failure recovery protocol as follows: When a node, x, locates a qualified substitute for a hole in $x.table$ using step (a), (b), (c), or (d) of the failure recovery protocol, if the qualified substitute is an S-node, then x fills the hole with it and terminates the recovery process. However, if the qualified substitute is a T-node, x saves the T-node in a waiting list for the entry and continues the recovery process. Only when the recovery process terminates at the end of step (d) without locating any S-node as a qualified substitute, will x remove a T-node from the entry's waiting list to fill the hole (provided that the list is not empty). Also, because of Rule 1, when a node searches among its neighbors and reverse-neighbors to find a qualified substitute for a hole in its table, or in response to a query from another node, it does not select a T-node as long as there are S-nodes that are qualified.

Rule 1 extends the basic join protocol as follows: Consider a node, x, that discovers a new neighbor, y, for one of its table entries after receiving a message from another node. x can store y in the table entry, if the table entry is not full with K neighbors yet and y is an S-node, according to the following steps. First, x checks if there exists any vacancy among the K "slots" of the entry that is not a hole for which failure recovery is in progress. If there exists such a vacancy, y is filled into it; otherwise, y (an S-node) is filled into a hole in the entry and the recovery process for the hole is terminated. On the other hand, if the new neighbor y is still a T-node, then y can be stored in the entry if the total number of neighbors and holes in the entry is less than K. Otherwise, y (a T-node) is saved in the entry's waiting list and may be stored into the entry later when the recovery process of a hole in the entry terminates.

Next, we present more extensions to the join protocol, presented as Rules 2-7. Rule 2 applies to both S-nodes and T-nodes, while Rules 3-7 apply to T-nodes only.

Rule 2 A node cannot reply to *CpRstMsg*, *JoinWaitMsg* or *JoinNotiMsg*, if the node has any ongoing recovery process at the time it receives such a message.

When a node, x, receives a *CpRstMsg*, *JoinWaitMsg* or *JoinNotiMsg*, if x has at least one recovery process that has not terminated, x needs to save the message and process it later. Each time a recovery process terminates, x checks whether there is any more recovery process still running. If not, x can process the above three types of messages it has saved so far.

Rule 3 When a T-node detects failure of a neighbor in its table, it starts a failure recovery process for each hole left by the failed neighbor with the following exception, which requires backtracking by the T-node.

Consider a T-node, say x. In order to backtrack, x keeps a list of nodes, (g_0, ..., g_i) to which it has sent a *CpRstMsg* or a *JoinWaitMsg*, in order of sending times. Backtracking is required if failure of the last node on the list, g_i, is detected under one of the following conditions: (i) when x is in status *copying* and waiting for a *CpRlyMsg* from g_i, (ii) when x is in status *waiting* and waiting for a *JoinWaitRlyMsg* from g_i, or (iii) when x is in status *notifying* and node g_i is currently the only reverse-neighbor of x (g_i must be a node that has replied positively to a *JoinWaitMsg* from x).

Note that under conditions (i) and (ii), x has not been attached to the network. Under condition (iii), failure of g_i may cause x to be detached from the network. In each case, x backtracks by deleting g_i (which has failed) from its table, setting its status to *waiting*, and sending a *JoinWaitMsg* to g_{i-1}, with g_i included in the message, to inform g_{i-1} about the failure of g_i and request g_{i-1} to store x into $g_{i-1}.table$. If g_{i-1} has also failed, then x contacts g_{i-2}, and so on. If x backtracks to g_0 and g_0 has also failed, then x has to obtain another node from the network to start joining from the beginning again.

Rule 4 A T-node must wait until its status is *notifying* before it can inform its neighbors, which will store it as a reverse-neighbor. (This is to prevent a T-node from being selected as a substitute for a hole before it is attached to the network.)

Rule 5 When a T-node receives a reply with a substitute for a hole in its table, if the T-node is in status *notifying* and the substitute node should be notified (see condition for sending out a *JoinNotiMsg* in Figure 15), then the T-node sends a *JoinNotiMsg* to the substitute node, even if the substitute node is not qualified to be filled into the hole.

Rule 6 A T-node cannot change status to *in_system* if it has any ongoing failure recovery process.

Rule 7 When a T-node changes status to *in_system*, it must inform all its neighbors, in addition to its reverse-neighbors, that it has become an S-node.

6.2 Simulation results

We implemented the extended join and failure recovery protocols and conducted 980 simulation experiments to evaluate them. Each simulation began with a K-consistent network, $\langle V, \mathcal{N}(V)\rangle$, of n nodes ($n = |V|$). Then a set W of nodes joined and a set F of nodes failed during the simulation. Each simulation was identified by a combination of b, d, K, n, and $|W| + |F|$ values, where $|W| + |F|$ is the total number of join and failure events. K was varied from 1 to 5, (b, d) values were chosen from (4,16),(4,64), (16,8) and (16,40), and three values, 1600, 3200 and 3600, were used for the initial network size (n). For 3200-node and 3600-node simulations, all joins and failures occurred at the same time. For 1600-node simulations, join and failure events were generated according to a Poisson process at the rate of 1 event per second in 220 experiments, 1 event every 10 seconds in 180 experiments, 1 event

every 20 seconds in 60 experiments, and 1 event every 100 seconds in 60 experiments. K-consistent neighbor tables for the initial network were constructed using the four approaches described in Section 5.

At the end of every simulation, we checked whether the join processes of all joining nodes that did not fail (nodes in $W - F$) terminated. We then checked whether the neighbor tables of all remaining nodes (nodes in $V \cup W - F$) satisfy K-consistency. Table 5 presents a summary of results of the 980 simulation experiments. We observed that, for $K \geq 2$, in *every* simulation, the join processes of all nodes in $W - F$ terminated and the neighbor tables of all remaining nodes satisfied K-consistency. Each such experiment is referred to in Table 5 as a simulation with perfect outcome.

		$K = 1$		$K = 2, 3, 4, 5$	
n	Num. of events ($\|W\| + \|F\|$)	Num. of sim.	Num. of sim. w/ perfect outcome	Num. of sim.	Num. of sim. w/ perfect outcome
1600	200 (38+162)	16	16	64	64
1600	200 (110+90)	16	16	64	64
1600	200 (160+40)	12	12	48	48
1600	400 (85+315)	12	10	48	48
1600	400 (204+196)	12	11	48	48
1600	400 (323+77)	12	12	48	48
1600	800 (386+414)	24	22	96	96
3600	400 (81+319)	16	13	64	64
3600	400 (210+190)	16	15	64	64
3600	400 (324+76)	12	12	48	48
3600	800 (169+631)	12	9	48	48
3600	800 (387+413)	12	11	48	48
3600	548 (400+148)	12	10	48	48
3200	1600 (780+820)	12	9	48	48

Table 5. Results for concurrent joins and failures

In the protocol extensions, the recovery process for a hole in a table has priority over a joining node in filling the hole. This tends to prevent a joining node from getting a low attach-level in the network, and thus from sending too many *JoinNotiMsg*. We conjecture that when joins and failures occur concurrently, the number of *JoinNotiMsg* sent by a joining node is comparable to that in a failure-free scenario. To validate the conjecture, we counted the number of *JoinNotiMsg* sent by each joining node in the simulation experiments. For each simulation, we calculated the average number of *JoinNotiMsg* sent by nodes in $W - F$, and then compared the average number with the upper bound on the expected number of *JoinNotiMsg* sent by a joining node in the absence of failures (for initial network size n and m joining nodes, such that $m = |W - F|$ and $n = |V - F|$). The upper bound is calculated according to Theorem 6. Table 6 presents results from 10 simulations for $|V| = 3600$, which demonstrates that the average numbers from simulations (for concurrent joins and failures) are comparable to the upper bounds from Theorem 6 (for concurrent joins in absence of failures).

7 K vs. Maintenance Cost

As shown in previous sections, the larger the K value in a K-consistent network, the more resilient is the network when nodes fail and the easier it is for the network to recover from failures and maintain K-consistency. However, these benefits come with a price. First, with a larger K, more neighbors are stored in each table. As a result, each node has to send more messages when it probes neighbors and exchanges information with neighbors. Also, with more neighbors in a table, the bigger a message would be if it includes a copy of the table.[15] Second, with a larger K, the overhead of a join to maintain K-consistency is higher. For example, a joining node needs to send more *JoinNotiMsg*.

$\|V\| = 3600$, $\|W\| = 387$, $\|F\| = 413$		$n = 3208$, $m = 366$, no failures	
K, b, d	Average number	K, b, d	Upper bound
1, 4, 64	2.519125	1, 4, 64	2.399764
2, 4, 64	4.800546	2, 4, 64	5.501709
3, 4, 64	6.808743	3, 4, 64	8.952706
4, 4, 64	9.696721	4, 4, 64	12.253937
5, 4, 64	12.193989	5, 4, 64	14.925359
1, 16, 40	7.390710	1, 16, 40	6.814094
2, 16, 40	11.740437	2, 16, 40	11.842158
3, 16, 40	14.404371	3, 16, 40	13.861656
4, 16, 40	15.382513	4, 16, 40	14.633307
5, 16, 40	15.128415	5, 16, 40	15.553342

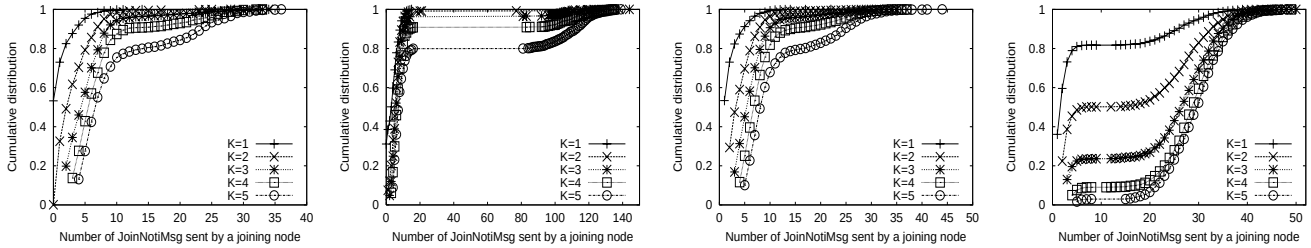
Table 6. Average number of JoinNotiMsg with failures vs. without failures (upper bound)

We first study the storage cost for maintaining K-consistency. The number of neighbors stored in a node's table is used as a measure of storage cost. We ran simulations for different combinations of K, b, d, and n values to calculate the average number of neighbors per node. In each simulation, neighbor tables were constructed according to Definition 3.3. Then the number of neighbors in each node's table was counted.[16] For each combination of parameter values, we ran five simulations to obtain the average number of neighbors per node. The results are shown in Figure 9, which shows that the average number of neighbors in a node's table depends on the values of b, K and n, but not on the value of d. In Figure 9, in decreasing order of the average number of neighbors, the curves are for $b = 16$ and $n = 5000$, $b = 16$ and $n = 1000$, $b = 4$ and $n = 5000$, and $b = 4$ and $n = 1000$. Note that the average number of neighbors per node is approximately $bK \log_b n$, given that node IDs are uniformly distributed over the ID space. Similarly, the average number of reverse-neighbors maintained by each node is also approximately $bK \log_b n$.

Next, we study the communication overhead of maintaining K-consistency. Simulation results in Section 5 show that for $K \geq 2$, most of recoverable holes can be recovered within the first two steps, where the first step imposes no overhead and the second step imposes at most $2(K - 1)$ messages. Therefore, maintenance overhead tends to be dominated by the overhead of joins. Next, we evaluate the communication overhead of a join for different K values in the absence of failures, since from simulation results in Section 6, we observed that the number of messages sent by a joining node in the presence of fail-

[15] In [7], we discussed how to reduce the size of messages that include a copy of a table.

[16] The node itself is not included in the number, but a neighbor stored in different entries of the table is counted multiple times. As a result, the total number of neighbors per node does not depend on how the neighbors in each entry are chosen from the set of qualified nodes in the network.



(a) b=4, d=64, n=1600, m=400 (b) b=16, d=40, n=1600, m=400 (c) b=4, d=64, n=6400, m=1600 (d) b=16, d=40, n=6400, m=1600

Figure 10. Cumulative distribution of the number of JoinNotiMsg sent by a joining node

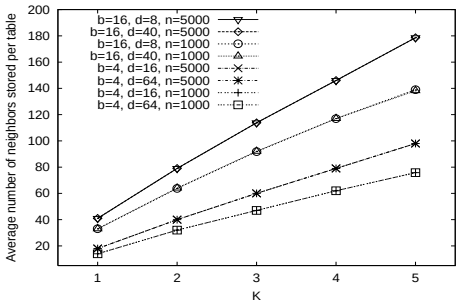


Figure 9. Average number of neighbors per node

ures is comparable to that in the absence of failures.

From Theorem 4, the combined number of messages sent by a joining node in status *copying* and status *waiting* is at most $d + 1$, which does not depend on K. However, the size of the reply to each of these messages becomes larger as K becomes larger, because each reply includes a neighbor table. In Section 4.4, we presented an upper bound of the expected number of *JoinNotiMsg*, and found that the larger the K, the higher the upper bound. We next study the distribution of the number of *JoinNotiMsg* sent by a joining node from simulation results. In each simulation, initially the network consisted of n nodes with K-consistent neighbor tables, where $n \in \{1600, 6400\}$ and $K \in \{1, ..., 5\}$. Then m nodes joined the network, $m = n/8$ or $m = n/4$. The number of *JoinNotiMsg* sent by each node during the simulation was counted and logged. For each combination of parameter values, we ran five simulations, each time with a different seed for random number generation. The distribution of the number of *JoinNotiMsg* sent by a joining node was computed from results of the five runs. Figure 10 presents distributions of the number of *JoinNotiMsg* for different values of K,[17] which shows that the larger the K value, the smaller is the percentage of nodes that sent a given number of *JoinNotiMsg*. For the other types of join protocol messages, the number of them sent by a joining node may also increase with K but these are small messages (which do not contain a neighbor table). Results of analysis of these small messages are presented in Appendix B. In general, the larger the K value, the more of each type of small messages are sent.

[17]We observed from the results that the distributions are insensitive to the value of m when $n \gg m$. Therefore we only present distributions for $m = n/4$.

8 Conclusions

We generalized the concept of consistency to K-consistency ($K \geq 1$) to improve the robustness of a hypercube routing scheme that is used in several proposed peer-to-peer networks. We showed that a K-consistent network provides at least K disjoint paths to every source-destination pair with probability close to 1. We then presented a generalized definition of C-set trees for protocol design and reasoning about K-consistency. We designed and specified a new join protocol together with a proof that it generates K-consistent neighbor tables for an arbitrary number of concurrent joins under the assumption that there is no concurrent leave or failure.

We next designed and evaluated a failure recovery protocol based upon local information. Extensions to the basic join and failure recovery protocols to handle concurrent joins and failures were then presented. For an initially K-consistent network, the impact of concurrent joins and failures was studied in a large number of simulation experiments. We found that in every experiment, for $K \geq 2$, our protocols constructed and maintained K-consistent tables after the joins and failures. These extended protocols are being implemented in our prototype system named Silk.

An observation from our study is that networks in which each node maintains a large number of consistent neighbor pointers are not only more resilient, but they also *recover more quickly and completely* from node failures than networks in which each node maintains a small number of consistent neighbor pointers. From our analytic and simulation results in Sections 3 to 6, we found that the improvement in network resilience from $K = 1$ to $K \geq 2$ is dramatic. We conclude that hypercube routing networks should be K-consistent with $K \geq 2$. Figure 10 shows that for $K \leq 4$, most joining nodes send a fairly small number of *JoinNotiMsg*. Therefore, we recommend choosing a value of K in the range of [2,4].

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*, 2002.

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

[3] D. R. Karger and M. Ruhl. Finding nearest neighbors in growth-restricted metrics. In *Proc. of ACM Symposium on Theory of Computing*, 2002.
[4] S. S. Lam and A. U. Shankar. A theory of interfaces and modules I–composition theorem. *IEEE Transactions on Software Engineering*, January 1994.
[5] X. Li and C. G. Plaxton. On name resolution in peer-to-peer networks. In *Proc. of the 2nd Workshop on Principles of Mobile Computing*, 2002.
[6] 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, Sept. 2002.
[7] 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*, 2003.
[8] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. of ACM Symposium on Parallel Algorithms and Architectures*, 1997.
[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of ACM SIGCOMM*, 2001.
[10] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Proc. of IEEE INFOCOM*, 2002.
[11] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of IFIP/ACM International Conference on Distributed Systems Platforms*, 2001.
[12] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of ACM SIGCOMM*, 2001.
[13] E. W. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proc. of IEEE Infocom*, 1996.
[14] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, Aug. 2001.

A Pseudocode for Join Protocol in Section 4

We have presented the state variables of a joining node in Figure 6 and protocol messages in Figure 7. Variables in the top part in Figure 6 are also used by nodes in V, the nodes in the initial network, where for each node u, $u \in V$, $u.status$ = *in_system*, $u.table$ is populated in a way that satisfies conditions in Definition 3.3, and $u.state(v) = S$ for every neighbor v that is stored in $u.table$. Figures 11 to 16 present a pseudocode specification of the protocol, in which x, y, u and v denote nodes, and h, i, j and k denote integers.

When any node, x, stores y into $N_x(i,j)$, $y \neq x$, x needs to sends a *RvNghNotiMsg*$(y, x.state(y))$ to y, 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. We also omit the sending of a *CpRstMsg* from x to g, and the reception of a *CpRlyMsg* from g to x, in Figure 11.

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, $v \in N_g(i,j)$) {
        $k' = |csuf(x.ID, v.ID)|$;
        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 11. **Action in status *copying***

B Proofs of Theorems

In this section, we present our proofs for Lemmas 3.1 and 3.2, and the proofs for Theorem 2 to 6.

B.1 Proofs of Lemmas 3.1, 3.2

Proof of Lemma 3.1: 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: $P_i = \{u_i^0, ..., u_i^k\}$, $i \in [K]$, $1 \leq k \leq d$, where $u_i^0 = x$, $u_i^k = y$, and $u_i^j \in N_{u_i^{j-1}}(j-1, y[j-1])$, $1 \leq j \leq k$. 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, $0 \leq l \leq i-1$, 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 $I \leq K$. 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, $h \leq i \leq I-1$, 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 $f \geq I$, then starting with P_h, for each path P_i, $h \leq i \leq I-1$, let $u_i^{j+1} = v$, such that $v \in N_{u_i^j}(j, y[j])$ and

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 12. **Action on receiving JoinWaitMsg and JoinWaitRlyMsg**

$v \neq u_l^{j+1}$ for all l, $0 \leq l \leq i-1$. 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, $h \leq i \leq f-1$, 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, $0 \leq l \leq i-1$, and (ii) for each path P_i, $f \leq i \leq I-1$, let $u_i^{j+1} = y$, because $f < I$ indicates $f < K$, i.e., $|V_{y[j]\ldots y[0]}| < K$, so every node in $V_{y[j]\ldots y[0]}$, 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 to y (see Lemma 3.1 in [7]).

So far we have established K paths from x to y. We then prove that they are disjoint. We need to prove the following claim first:

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

Proof: Prove by induction. Base step: by the way we assign nodes to $u_{i'}^1$ for each path $P_{i'}$, we know that $u_i^1 \neq u_l^1$.

Inductive step: Suppose $u_i^j \neq u_l^j$, $j \geq 1$, 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., $u_l^{j+1} \neq u_i^{j+1}$.

■

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 ($j = i$; $j \leq k$, j++){ Set_Neighbor($j, x[j], x, T$);}
for ($j = i$; $j \leq k$, 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 13. **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==ϕ and Q_{sr}==ϕ) Switch_To_S_Node();

Figure 14. **Action on receiving SpeNotiMsg and SpeNotiRlyMsg**

Second, by Claim B.1, we can show that 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 $y.ID$, hence, $x[0] = y[0]$ and $x \in N_x(0, y[0])$. Hence, there must exist a path P_l such that $u_l^1 = x$ (by 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, $u_l^1 = ... = u_l^j = u_l^{j+1} = x$. Next, by Claim B.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 $u_i^{j+1} = x$.

Third, we point out that the K paths are different from each other, since in each path, at least u_i^1 is different from each other.

Check_Ngh_Table(y.table) at x:

for (each $N_y(i,j), i \in [d], j \in [d]$) {
 for (each u, $u \in N_y(i,j) \wedge u \neq x$) {
  $k = |csuf(x.ID, u.ID)|$; $s = y.state(u)$;
  for ($h = i$; $h \leq k$; h++) { Set_Neighbor($h, u[h], u, s$); }
  if ($x.status$ == *notifying* $\wedge\ k \geq x.att\_level \wedge u \notin Q_n$) {
   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 ($j \leq k \wedge h == -1$){
  if (for each l, $j \leq l \leq k$, $N_x(l, u[l]).size < K$){
   $h = j$; **for** ($l = h$; $l \leq k$; 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 15. **Subroutines**

Action of y on receiving a InSysNotiMsg from x:

$y.state(x) = S$;

Figure 16. **Action on receiving InSysNotiMsg**

Based on the above results, we prove that the K paths are disjoint. Consider any two paths P_i and P_l. By Claim B.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 3.2: (Outline) By Lemma 3.1, 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 3.1, 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, $N_x(0, x[0]).size = \min(K, |V_{y[0]}|)$. Similar to the proof for Lemma 3.1, 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 3.1, 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.2 Correctness of join protocol

In this section, we present proofs for Theorems 2 and 3. 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 by Figures 11, 12, 13, 14, and 16 are atomic.

Theorem 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, at time t^e, $\langle V \cup W, \mathcal{N}(V \cup W)\rangle$ is a K-consistent network.*

To prove Theorem 2, we first prove some auxilary lemmas and propositions. We start by presenting some definitions. Recall that V_x^{Notify}, the notification set of x regarding V, is defined in Definition 4.1.

Definition B.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. The period from t_x^b to t_x^e, denoted by $[t_x^b, t_x^e]$, is the* **joining period** *of x.*

Definition B.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 B.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 B.4 *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 B.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* **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* $V_y^{Notify} \subset V_u^{Notify}$.

Notation	*Definition*
$\langle x \rightarrow y \rangle_k$	x can reach y within k hops
$x \stackrel{j}{\rightarrow} y$	the action that x sends a *JN* or a *JW* to y
$x \stackrel{jn}{\rightarrow} y$	the action that x sends a *JN* to y
$x \stackrel{jw}{\rightarrow} y$	the action that x sends a *JW* to y
$x \stackrel{c}{\rightarrow} y$	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 7. Notation in proofs

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

Table 8. Abbreviations for protocol messages

Table 7 presents the notation used in the following proofs, while Table 8 shows the abbreviations we will use for protocol messages in the proofs.

The following facts, which are easily oberved from the join protocol, are used frequently in the proofs.

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 a* JW *from x. Also, x changes status from waiting to notifying immediately after it receives a* 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 $x \stackrel{jn}{\rightarrow} y$ 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_x^e, 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 $y \neq x$:*

- *$x \stackrel{jn}{\rightarrow} y$ 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 $x \stackrel{j}{\rightarrow} z$ happens before time t_x^e.

Proof: See the proof of **Proposition A.1** in [6]. (Note that in this report, we rename the variable $x.noti\_level$ in [6] as $x.att\_level$.) ■

Lemma B.1 *Suppose node x joins a K-consistent network $\langle V, \mathcal{N}(V) \rangle$. Then, at time t_x^e, $\langle V \cup \{x\}, \mathcal{N}(V \cup \{x\}) \rangle$ is a K-consistent network.*

Proof: Suppose $V_x^{Notify} = V_{x[k-1]...x[0]}$, $k \in [d]$, 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 that the condition for x to change status is that x finds there exists a level-h in the table of g, such that $N_g(i, x[i]).size < K$, $h \leq i \leq |csuf(x.ID, g.ID)|$. And since $V_{x[k-1]...x[0]} \geq K$, $V_{x[k]...x[0]} \geq K$, 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, $x \stackrel{j}{\rightarrow} z$ 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 $|V_{j\cdot x[i-1]\ldots x[0]}| \geq K$; and $N_x(i,j) \supseteq V_{j\cdot x[i-1]\ldots x[0]}$ if $|V_{j\cdot x[i-1]\ldots x[0]}| < K$.

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 B.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 B.1, Lemma B.2 holds when $i = 1$. Assume when $1 \leq i < m$, Lemma B.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 B.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 B.2 holds for $i+1$. ■

Lemma B.3 *Suppose a set of nodes, $W = \{x_1, ..., x_m\}$, $m \geq 2$, join a K-consistent network $\langle V, \mathcal{N}(V)\rangle$ independently. For any node x, $x \in W$, if $|V_{j\cdot x[i-1]\ldots x[0]}| < K$, $0 \leq i < d-1$, $j \in [b]$, then $(V \cup W')_{j\cdot x[i-1]\ldots x[0]} = V_{j\cdot x[i-1]\ldots x[0]}$, where $W' \subseteq W - \{x\}$.*

Proof: Similar to the proof of **Lemma A.3** in [6]. ■

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, $x \in G(V_{\omega_1})$, $(V \cup G(V_{\omega_2}))_{j\cdot x[i-1]\ldots x[0]} = V_{j\cdot x[i-1]\ldots x[0]}$ if $|V_{j\cdot x[i-1]\ldots x[0]}| < K$.*

Lemma B.4 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, join a K-consistent network $\langle V, \mathcal{N}(V)\rangle$ concurrently. If the joins are independent, then at time t^e, $\langle V \cup W, \mathcal{N}(V \cup W)\rangle$ is a K-consistent network.*

Proof: Consider any node x, $x \in W$. If $|V_{j\cdot x[i-1]\ldots x[0]}| \geq K$, then by Corollary B.1, by time t^e, $N_x(i,j).size = K$. If $|V_{j\cdot x[i-1]\ldots x[0]}| < K$, then by Lemma B.3, we have $(V \cup W)_{j\cdot x[i-1]\ldots x[0]} = V_{j\cdot x[i-1]\ldots x[0]}$ for $j \neq x[i]$, and $(V \cup W)_{j\cdot x[i-1]\ldots x[0]} = V_{j\cdot x[i-1]\ldots x[0]} \cup \{x\}$ for $j = x[i]$, $i \in [d]$ and $j \in [b]$. Then, by Corollary B.1, $N_x(i,j).size = |V_{j\cdot x[i-1]\ldots x[0]}|$ for $j \neq x[i]$; and $N_x(i,j).size = |V_{j\cdot x[i-1]\ldots x[0]}| + 1$ for $j = x[i]$, where $N_x(i,j) = V_{j\cdot x[i-1]\ldots x[0]} \cup \{x\}$. 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 $|V_{j\cdot y[i-1]\ldots y[0]}| \geq K$, then $N_y(i,j).size = K$ since the initial network is K-consistent. If $|V_{j\cdot y[i-1]\ldots y[0]}| < K$ and $W_{j\cdot y[i-1]\ldots y[0]} = \emptyset$, then $N_y(i,j) = V_{j\cdot y[i-1]\ldots y[0]} = (V \cup W)_{j\cdot y[i-1]\ldots y[0]}$. If $|V_{j\cdot y[i-1]\ldots y[0]}| < K$ and $W_{j\cdot y[i-1]\ldots y[0]} \neq \emptyset$, 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.3, x is the only node in W has the suffix $j \cdot y[i-1]...y[0]$. Similar to the argument in proving Lemma B.1, we can prove that $x \xrightarrow{j} y$ happens before time t^e_x. Hence, $N_y(i,j) = V_{j\cdot y[i-1]\ldots y[0]} \cup \{x\} = (V \cup W)_{j\cdot y[i-1]\ldots y[0]}$.

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. ■

Lemma B.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. If the joins are dependent, then at time t^e, $\langle V \cup W, \mathcal{N}(V \cup W)\rangle$ is K-consistent.*

We prove Lemma B.5 by induction upon the C-set trees realized by the end of all joins (see Definition 4.3). We first consider the case where all joining nodes in W belong to the same C-set tree and prove that eventually, neighbor tables of these joining nodes as well as the nodes that are initially in the network satisfy K-consistency conditions (stated in Proposition B.12). Then, we prove Proposition B.13, which states when joining nodes belong to different C-set trees, their neighbor tables eventually satisfy K-consistency conditions. Based on Proposition B.12 and Proposition B.13, we present our proof of Lemma B.5.

In the following proofs, we define $l_j...l_1$ to be the empty string if $j = 0$. Also, we define **the first C-set x belongs to** for a node x, $x \in W$, to be (i) $C_{l_1\cdot\omega}$ if $x \in C_{l_1\cdot\omega}$; (ii) $C_{l_j\ldots l_1\cdot\omega}$ for $j > 1$, if $x \in C_{l_j\ldots l_1\cdot\omega}$ and $x \notin C_{l_{j-1}\ldots l_1\cdot\omega}$.

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^e_x, $\langle y \rightarrow x\rangle_d$.*

Proof: Similar to the proof of **Proposition A.2** in [6], by replacing "$N_{u_i}(h_i, x[h_i]) = null$" by "$N_{u_i}(h_i, x[h_i]).size < K$" in case (1), and replacing "$N_{u_i}(h_i, x[h_i]) = v$, $v \neq x$" by "$N_{u_i}(h_i, x[h_i]).size = K$, $y \notin N_{u_i}(h_i, x[h_i])$", and "let $u_{i+1} = v$" by "let $u_{i+1} = N_{u_i}(h_i, x[h_i]).first$" in case (2). ■

Proposition B.3 *Suppose a set of nodes, $W = \{x_1, ..., x_m\}$, $m \geq 2$, join a K-consistent network $\langle V, \mathcal{N}(V)\rangle$. 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)* [6].[18] 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.2 *(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.2: **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 arguemnt 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 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.

Case 2: $|csuf(u.ID, x.ID)| < h$. Proof in this case is similar to that in Case 3 in the proof of **Proposition A.3** in [6], by replacing notation in the form of "$N_u(i, z[i]) = z$" by "$z \in N_u(h', x[h'])$", "$v_{i+1} = N_{v_i}(h_i, x[h_i])$" by "$v_{i+1} = N_{v_i}(h_i, x[h_i]).first$", and "$N_u(h', x[h']) = v$, where $v \neq x$ and $v \neq y$" by "$y \notin N_u(h', x[h'])$". ■

Proofs of the following propositions and corollaries are based on induction upon C-set trees. Propositions B.4 to B.12 assume that all joining nodes belong to the same C-set tree, which is the same assumption as made in Definition 4.3, namely:

Assumption B.1 *(for Propositions B.4 to B.12)*
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, $x \in W$, $V_x^{Notify} = V_\omega$, $|\omega| = k$.

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

Proof: Consider *contact-chain(x,g)*, where g is the node that x is given to start its join process. Suppose *contact-chan(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* [6]). 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 B.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 B.5 *If $|V_{l_j...l_1\cdot\omega}| < K$ and $W_{l_j...l_1\cdot\omega} \neq \emptyset$, $1 \leq j \leq d-k$, $l_j, ..., l_1 \in [b]$, then*

(a) $C_{l_j...l_1\cdot\omega} \supseteq V_{l_j...l_1\cdot\omega}$;
(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

[18] The definition of *contact-chain(y,u)* in a K-consistent network is similar to what is presented in the proof of **Proposition A.3** in [6], by replacing "$N_{u_i}(h_i, y[h_i]) = y$" by "$y \in N_{u_i}(h_i, y[h_i])$", "$u$ has set $N_{u_i}(h_i, y[h_i]) = v, v \neq y$ by "$|N_{u_i}(h_i, y[h_i])| = K$, $y \notin N_{u_i}(h_i, y[h_i])$, and $v = N_{u_i}(h_i, y[h_i]).first$".

$x \in W_{l_j...l_1\cdot\omega}$ 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 4.3, 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', $i + 1 \le i' \le d - k$, where $l_{i'}...l_1 \cdot \omega$ is a suffix of $x.ID$. Therefore, it must be that $i + 1 > j$, i.e., $i \ge j$ (otherwise, $x \in C_{l_j...l_1\cdot\omega}$). However, by Corollary B.5, $|C_{l_{j'}...l_1\cdot\omega}| \ge K$ for $1 \le j' \le i$, thus, $|C_{l_j...l_1\cdot\omega}| \ge K$. A contradiction. ■

Proposition B.6 *Consider any node x, $x \in W$, if $x \in C_{l_{j+1}...l_1\cdot\omega}$ and $x \notin C_{l_j...l_1\cdot\omega}$, $1 \le j \le d - k - 1$, (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 $u.table$, $0 \le i \le d-k-1$. 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 $i \ge j$. Otherwise, if $i < j$, then since $x \in C_{l_{i+1}...l_1\cdot\omega}$, it follows that $x \in C_{l_{i'}...l_1\cdot\omega}$, $i' \le i \le d - k$, thus $x \in C_{l_j...l_1\cdot\omega}$, which contradicts with the assumption in the proposition.

Next, we show that $i \le j$, 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 13 and 15.) 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 $y \ne x$, 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 14), thus when y copies x from $z.table$, it can not fill x into a level lower than $i + k$ (again, see Figure 15). 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 $i \le j$.

Since $i \ge j$ and $i \le j$, 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, $2 \le j \le d - k$, then $|C_{l_i...l_1\cdot\omega}| \ge K$ for $1 \le i < j$.*

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 B.4. Thus, $g_j[i' - 1]...g_0[0] = x[i' - 1]...x[0]$, $0 \le i' \le h$. 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 4.3), and hence $N_{g_{k+i}}(h', x[h']).size < K$, where $k+i \le h' \le |csuf(x.ID, g_{k+i}.ID)|$. 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.7 *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, $1 \le j \le d - k$. 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}, $t_{xy} = \max(t^e_x, t^e_y)$, $\langle y \to x\rangle_d$.*

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.6, $u_y \in C_{l_{j-1}...l_1\cdot\omega}$ (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, $\langle y \to x\rangle_d$ 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, $\langle y \to x\rangle_d$ by time t^e_y. (ii) Otherwise, $y \xrightarrow{j} x$ eventually happens ($|csuf(x.ID, y.ID)| \ge k + j > y.att\_level$).

- 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.
- 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 13). 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 14) 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 (ii) in the last part of the proof of Proposition B.7. ■

Proposition B.8 *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) $N_x(k + j - 1, l).size = \min(K, |(V \cup W)_{l \cdot l_{j-1} ... l_1 \cdot \omega}|)$ 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.6, $y.att\_level \leq j + k - 1$ 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 B.5, $C_{l \cdot l_{j-1} ... l_1 \cdot \omega} = (V \cup W)_{l \cdot l_{j-1} ... l_1 \cdot \omega}$ if $|(V \cup W)_{l \cdot l_{j-1} ... l_1 \cdot \omega}| < K$, and $|C_{l \cdot l_{j-1} ... l_1 \cdot \omega}| \geq K$ if $|(V \cup W)_{l \cdot l_{j-1} ... l_1 \cdot \omega}| \geq K$. ■

Proposition B.9 *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 assertions hold:*

(a) If $|W_{l_j ... l_1 \cdot \omega}| \geq 2$, then for any two nodes, x and y, where $x \in C_{l_j ... l_1 \cdot \omega}$, $y \in C_{l_j ... l_1 \cdot \omega}$, $x \neq y$, 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, $x \in C_{l_j ... l_1 \cdot \omega}$ and $x \in W$, $N_x(k + j - 1, l).size = \min(K, |(V \cup W)_{l \cdot l_{j-1} ... l_1 \cdot \omega}|)$ by time t^e.

Proof: We prove the proposition by induction on j. The structure of the proof is very similar to that of **Proposition A.6** in [6].

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.6, 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.6, $x.att\_level = y.att\_level = k$. Therefore, both $x \xrightarrow{j} u_x$ and $y \xrightarrow{j} u_y$ happens. Also, by part(a) of Proposition B.8, $x \xrightarrow{j} u_y$ happens. Likewise, $y \xrightarrow{j} u_x$ happens. By Proposition B.7, $\langle y \rightarrow x \rangle_d$ 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 $N_{u_y}(k, l)$.

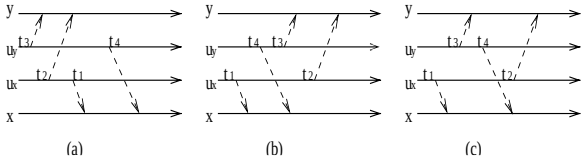


Figure 17. Message sequence chart for base case

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 $j+1$, $1 \le j \le d-k-1$.

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.3 *Suppose Proposition B.9 holds at j, $1 \le j \le d-k-1$. 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 may or may not be equal to l_{j+1}, 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.*

The proof of Claim B.3 is similar to that of **Claim A.2** in [6] and is omitted here. (Note that both x and y are in W.)

We next prove the proposition case by case.

Case 1.a.1. By Proposition B.6, 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.3, $y \xrightarrow{j} x$ 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.7, $\langle y \to x\rangle_d$.

(ii) $u_x \in V$ and $u_y \in W$. By Proposition B.8, $u_y \xrightarrow{j} u_x$ happens. Let t_a be the time that u_x receives the notification from u_y.

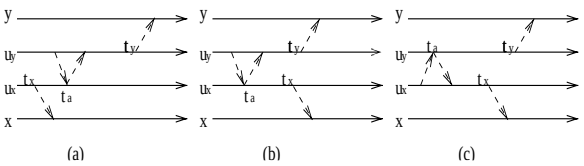


Figure 18. Message sequence chart for case 1.a.1

(1) Suppose $t_x < t_a$, as shown in Figure 18(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.7, $\langle y \to x\rangle_d$.

(2) Suppose $t_x > t_a$, as shown in Figure 18(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 $|csuf(u_y.ID, x.ID)| \ge l+k = x.att\_level$ (see Fact B.5). Hence, $x \xrightarrow{j} u_y$ happens. By Proposition B.7, $\langle y \to x\rangle_d$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 Since $u_y.att\_level \le l-1+k$ and $|csuf(v.ID, u_y.ID)| > h \ge l+k$, $u_y \xrightarrow{j} v$ 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 $|csuf(v.ID, u_y.ID)| \ge l+k$. Then, by Proposition B.3, 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 $t_y \ge t^e_{u_y}$ by Fact B.2). Thus, $x \in N_{u_y}(l+k, l_{j+1})$ at time t_{xu_y}. By Proposition B.7, $\langle y \to x\rangle_d$ 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 18(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, obeserve 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 19(a). If $u_x \in V$, then by Proposition B.8, $y \xrightarrow{j} u_x$ happens by t_y^e. 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.

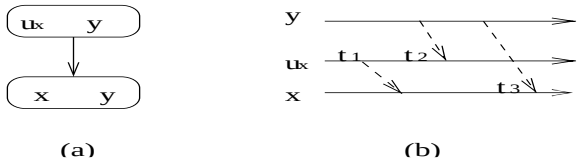


Figure 19. Message sequence chart for case 1.a.2

(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 not send out notifications), as shown in Figure 19(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_y^e. 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.7, $\langle x \rightarrow y \rangle_d$. 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.9 holds at j, part(a) holds in case 2.a.2 trivially.

So far, we have proved that part (a) of Proposition B.9 holds. Next, we prove part (b).

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 $y \in W$. If $y \notin C_{l_j \ldots l_1 \cdot \omega}$ (Case 1.b.1), then by Claim B.3, 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. 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.8. 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 B.5, $|C_{l \cdot l_j \ldots l_1 \cdot \omega}| = \min(K, |(V \cup W)_{l \cdot l_j \ldots l_1 \cdot \omega}|)$. 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.8, $x \xrightarrow{j} u_y$ 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 B.5 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_x^e, $y \in N_x(j-1+k,l)$ *or* $N_x(j-1+k,l).size = K$.

Proof: Proof of the corollary is implied by the proof of Proposition B.9. If $x \in V$, then by Proposition B.8, 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.9, 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_x^e. 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.3, 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_x^e. Hence, either $y \in N_x(j-1+k,l)$ or $N_x(j-1+k,l).size = K$ by t_x^e.

  Second, suppose $y \in C_{l_{j-1} \ldots l_1 \cdot \omega}$. By the proof of Case 1.b in proving Proposition B.9, 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 $u_x = A(x)$.
- $x \in C_{l_{j-1} \ldots l_1 \cdot \omega}$. Again, consider any node y, $y \in C_{l \cdot l_{j-1} \ldots l_1 \cdot \omega}$. First, suppose $y \in C_{l_{j-1} \ldots l_1 \cdot \omega}$, then both x and y belong to $C_{l_{j-1} \ldots l_1 \cdot \omega}$. By part(a) of Proposition B.9, 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 $y \notin C_{l \cdot l_{j-1} \ldots l_1 \cdot \omega}$. By the proof of Case 2.b in proving Proposition B.9, 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 B.10 *For any* x, $x \in W$, *suppose* $C_{l_j \ldots l_1 \cdot \omega}$ *is the first C-set* x *belongs to,* $1 \leq j \leq d-k$, l_1,...,$l_j \in [b]$. *Then* $N_x(k+i,l).size = \min(K, |(V \cup W)_{l \cdot l_i \ldots l_1 \cdot \omega}|)$ *for* $0 \leq i \leq j$ *and* $l \in [b]$.

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 $(u_0, u_1, \ldots u_f, u_{f+1})$, where $u_0 = g$, u_f is the node that sends an positive *JWRly* to x (see Definition of a *contact-chain*, in [6]) 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.6), then $k \leq j \leq |csuf(u.ID, x.ID)|$ (recall k is defined in Assumption B.1). Create a new sequence $(g_0, ..., g_j)$ as described in the proof of Proposition B.4, such that $g_0 = g$, $g_j = u_f$, and $g_{i'}.ID$ shares suffix $x[i'-1]...x[0]$ with $x.ID$, $0 \leq i' \leq j$. Then, it is easy to check that $g_k \in V_\omega$, and $g_{i'+k} \in C_{l_{i'} \ldots l_1 \cdot \omega}$, $1 \leq i' \leq j$. Thus, $g_{i+k} \in C_{l_i \ldots l_1 \cdot \omega}$. 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 $|(V \cup W)_{l \cdot l_i \ldots l_1 \cdot \omega}| < K$, then by Proposition B.5, $C_{l \cdot l_i \ldots l_1 \cdot \omega} = (V \cup W)_{l \cdot l_i \ldots l_1 \cdot \omega}$, i.e., for each y, $y \in W_{l \cdot l_i \ldots l_1 \cdot \omega}$, $y \in C_{l \cdot l_i \ldots l_1 \cdot \omega}$. Next, consider any node y, $y \in W_{l \cdot l_i \ldots l_1 \cdot \omega}$. 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_y^e; if $g_{i+k} \in V$, then $y \xrightarrow{j} g_{i+k}$ eventually happens by Proposition B.8. (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.3, at least one of the following is true: by time t_x^e, $y \in N_x(i-1+k,l)$, or that $N_x(i-1+k,l).size = K$. Since this conclusion is true for each y, $y \in C_{l \cdot l_i \ldots l_1 \cdot \omega}$, plus that $V_{l \cdot l_i \ldots l_1 \cdot \omega} \subset N_x(i-1+k,l)$ by time t^e (by Corollary B.1), we conclude that $N_x(i-1+k,l).size = \min(K, |(V \cup W)_{l \cdot l_i \ldots l_1 \cdot \omega}|)$ by time t^e.

If $|(V \cup W)_{l \cdot l_i \ldots l_1 \cdot \omega}| \geq K$, then by Proposition B.5, $|C_{l \cdot l_i \ldots l_1 \cdot \omega}| \geq K$. Next, consider any node y, $y \in C_{l \cdot l_i \ldots l_1 \cdot \omega}$ 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.3, by time t_x^e, either $y \in N_x(i-1+k,l)$ or $N_x(i-1+k,l).size = K$. Therefore, for any y, $y \in C_{l \cdot l_i \ldots l_1 \cdot \omega}$, either that $y \in N_x(i-1+k,l)$ by time t_x^e, or $N_x(i-1+k,l).size = K$ by time t_x^e. Hence, $N_x(i-1+k,l).size = K$ by time t^e. ■

Proposition B.11 *For any node* x, $x \in W$, *if* $(V \cup W)_{l \cdot l_i \ldots l_1 \cdot \omega} \neq \emptyset$, *where* $l \in [b]$ *and* $l_i ... l_1 \cdot \omega$ *is a suffix of* $x.ID$, $0 \leq i \leq d-k$, *then* $N_x(i+k,l).size = \min(K, |(V \cup W)_{l \cdot l_i \ldots l_1 \cdot \omega}|)$ *by time* t^e.

Proof: If $(V \cup W)_{l \cdot l_i \ldots l_1 \cdot \omega} = V_{l \cdot l_i \ldots l_1 \cdot \omega}$, by Corollary B.1, the proposition holds. If $(V \cup W)_{l \cdot l_i \ldots l_1 \cdot \omega} \supset V_{l \cdot l_i \ldots l_1 \cdot \omega}$, then consider C-set $C_{l_j \ldots l_1 \cdot \omega}$. Suppose $C_{l_j \ldots l_1 \cdot \omega}$ is the first C-set x belongs to, $0 \leq j \leq d-k$. If $j > i$, by Proposition B.10, the proposition holds. If $j \leq i$, then by part(b) of Proposition B.9, the proposition holds. ■

Proposition B.12 *For each node x, $x \in V \cup W$, if $(V \cup W)_{j \cdot x[i-1]...x[0]} \neq \emptyset$, $i \in [d]$, $j \in [b]$, then $N_x(i+k,l).size = \min(K, |(V \cup W)_{j \cdot x[i-1]...x[0]}|)$ by time t^e.*

Proof: First, pick any node x, $x \in W$.

- If $0 \leq i < k$, then by Corollary B.1, the proposition holds.
- If $i = k$ and $|V_{j \cdot x[i-1]...x[0]}| \geq K$, 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 $k < i \leq d-1$, then by Proposition B.11, 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 B.1 ($V_x^{Notify} = V_\omega$ 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 $|V_\omega| \geq K$ by Assumption B.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.8, $N_y(i+k,l).size = \min(K, |(V \cup W)_{j \cdot y[i-1]...y[0]}|)$ by time t^e. The proposition holds.

■

Propositions B.4 to B.12 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 B.13 *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, $N_x(k_2,l).size = \min(K, |(V \cup W)_{l \cdot \omega_2}|)$ for any x, $x \in V \cup W$, such that $x.ID$ has suffix ω_2, where $l \in [b]$.*

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 the above claim 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 $V_z^{Notify} = V_{\omega_3}$).

Next, consider node x. If $x \in V$ and $x.ID$ has suffix ω_2, then $x \in V_{\omega_2}$. By part (b) of Proposition B.8 $N_x(k_2,l).size = \min(K, |(V \cup G(V_{\omega_2}))_{l \cdot \omega_2}|)$. Thus, $N_x(k_2,l).size = \min(K, |(V \cup W)_{l \cdot \omega_2}|)$.

If $x \in W$, then 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 B.4, 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 V_{ω_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 B.5 , such a node must exist). By Proposition B.6, 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, $v \xrightarrow{j} g_{k_2}$ eventually happens for each u, $u \in V_{\omega_1}$, since by the time u_v replies to v, $\langle u_v \rightarrow g_{k_2} \rangle_d$.

Then, by Proposition B.3, 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 $v \in W$. Also, by Corollary B.1, $V_{l \cdot \omega_2} \subset N_x(k_2,l)$. 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 B.5.

Proof of Lemma B.5: 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 ω, $\omega \in \Omega$, $|V_\omega| \geq K$ by Definition 4.1. 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.4 *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.4: 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 $W_{j \cdot x[i-1]...x[0]} \subseteq$

$G(V_{\omega_l})_{j\cdot x[i-1]\ldots x[0]}$. In other words, we need to show that for any node y, $y \in W_{j\cdot x[i-1]\ldots x[0]}$, $V_y^{Notify} = V_{\omega_l}$ (thus $y \in G(V_{\omega_l})_{j\cdot x[i-1]\ldots x[0]}$).

For any node y, $y \in W_{j\cdot x[i-1]\ldots x[0]}$, $j \cdot x[i-1]\ldots x[0]$ is a suffix of $y.ID$. Since ω_l is a suffix of $j \cdot x[i-1]\ldots x[0]$ and $\omega_l \neq j \cdot x[i-1]\ldots x[0]$, 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 $j \cdot x[i-1]\ldots x[0]$, since it is given $|V_{j\cdot x[i-1]\ldots x[0]}| < K$. 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 $j \cdot x[i-1]\ldots x[0]$, $|\omega_l| \geq |\omega_h|$. 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.8 $N_x(i,j).size = \min(K, |(V \cup G(V_{\omega_l}))_{j\cdot x[i-1]\ldots x[0]})$. Then, by Claim B.4, $N_x(i,j).size = \min(K, |(V \cup W)_{j\cdot x[i-1]\ldots x[0]}|)$ 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 $|V_{j\cdot x[i-1]\ldots x[0]}| \geq K$, then $N_x(i,j).size = K$ by Corollary B.1. (ii) If $|V_{j\cdot x[i-1]\ldots x[0]}| < K$ and $W_{j\cdot x[i-1]\ldots x[0]} = \emptyset$, then $N_x(i,j).size = |V_{j\cdot x[i-1]\ldots x[0]}| = |(V \cup W)_{j\cdot x[i-1]\ldots x[0]}|$, again, by Corollary B.1.

  (iii) If $|V_{j\cdot x[i-1]\ldots x[0]}| < K$ and $W_{j\cdot x[i-1]\ldots x[0]} \neq \emptyset$, then $j \cdot x[i-1]\ldots x[0] \notin \Omega$. Since both ω_f and $x[i-1]\ldots x[0]$ are suffixes of $x.ID$, we next consider two cases: ω_f is a suffix of $x[i-1]\ldots x[0]$ or vice versa. If ω_f is a suffix of $x[i-1]\ldots x[0]$, then for any node y, $y \in W_{j\cdot x[i-1]\ldots x[0]}$, $y \in G(V_{\omega_f})$ (that is, x and y are in the same C-set tree). By Proposition B.11, $N_x(i,j).size = \min(K, (V \cup G(V_{\omega_f}))_{j\cdot x[i-1]\ldots x[0]})$, thus $N_x(i,j).size = \min(K, (V \cup W)_{j\cdot x[i-1]\ldots x[0]})$.

  If $x[i-1]\ldots x[0]$ 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 $j \cdot x[i-1]\ldots x[0]$ among Ω. Then, by Claim B.4, for any node y, $y \in W_{j\cdot x[i-1]\ldots x[0]}$, $y \in G(V_{\omega_l})$ (x and y are in different C-set trees). Note that since $|V_{j\cdot x[i-1]\ldots x[0]}| < K$ and $|V_{\omega_l}| \geq K$, it is impossible that $j \cdot x[i-1]\ldots x[0] = \omega_l$. Hence, ω_l is a suffix of $x[i-1]\ldots x[0]$, which is a suffix ω_f. Therefore, ω_l is a suffix of ω_f, then by Proposition B.13, $N_x(i,j).size = \min(K, (V \cup G(V_{\omega_l}))_{j\cdot x[i-1]\ldots x[0]})$, thus $N_x(i,j).size = \min(K, (V \cup W)_{j\cdot x[i-1]\ldots x[0]})$.

■

Lemma B.6 *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: First, separate nodes in W into groups, such that joins of nodes in the same group are dependent and joins of nodes in different groups are mutually independent, as follows (initially, let $i = 1$):

- For each node y, $y \in W - \bigcup_{j=1}^{i} G_j$, if there exists a node x, $x \in G_i$, such that $(V_y^{Notify} \cap V_x^{Notify} \neq \emptyset)$ or $(\exists u, u \in W - \bigcup_{j=1}^{i-1} G_j$, $(V_y^{Notify} \subset V_u^{Notify}) \wedge (V_x^{Notify} \subset V_u^{Notify}))$, put y in G_i;
- Pick any node x', $x' \in W - \bigcup_{j=1}^{i} G_j$, put x' in G_{i+1}, increment i and repeat these two steps until there is no node left.[19]

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, $x \in G_i$, $1 \leq i \leq l$, and any node y, $y \in G_j$, $1 \leq j \leq l$ and $i \neq j$. That is, nodes in the same group join dependently, while nodes in different groups join independently.

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

If $|V_{j\cdot x[i-1]\ldots x[0]}| < K$ and $W_{j\cdot x[i-1]\ldots x[0]} \neq \emptyset$, then $|(V \cup W)_{j\cdot x[i-1]\ldots x[0]}| = |(V \cup G_f)_{j\cdot x[i-1]\ldots x[0]}|$, where $(G_f)_{j\cdot x[i-1]\ldots x[0]} \neq \emptyset$. By Lemma B.5, $N_x(i,j).size = \min(K, |(V \cup G_f)_{j\cdot x[i-1]\ldots x[0]}|)$, hence, $N_x(i,j).size = \min(K, |(V \cup W)_{j\cdot x[i-1]\ldots x[0]}|)$. ■

Proof of Theorem 2: If $m = 1$, then by Lemma B.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 $[t^b_{G_i}, t^e_{G_i}]$, $1 \leq i \leq l$, where $t^b_{G_i} = \min(t^b_x, x \in G_i)$ and $t^e_{G_i} = \max(t^e_x, x \in G_i)$. We number the groups in such a way that $t^e_{G_i} \leq t^b_{G_{i+1}}$. Then, if $|G_1| \geq 2$, by Lemma B.6, at time $t^e_{G_1}$, $\langle V \cup G_1, \mathcal{N}(V \cup G_1)\rangle$ is a K-consistent network; if $|G_1| = 1$, then by Lemma B.1, $\langle V \cup G_1, \mathcal{N}(V \cup G_1)\rangle$ is a K-consistent network at time $t^e_{G_1}$. Similarly, by applying Lemma B.6 (or Lemma B.1) to G_2, ..., G_l, we conclude that evetually, at time t^e, $\langle V \cup W, \mathcal{N}(V \cup W)\rangle$ is a K-consistent network. ■

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, each node x, $x \in W$, eventually becomes an S-node.*

Proof: Similar to the proof of **Theorem 2** in [6]. ■

s

B.3 Communication cost of join protocol

Theorem 4 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 1$, join a K-consistent network $\langle V, \mathcal{N}(V)\rangle$. Then, for any x, $x \in$*

[19] [6] presents an example of how to group nodes following the steps for $K = 1$. See **Footnote 16** in [6].

W, the total number of CpRstMsg and JoinWaitMsg sent by x is at most $d+1$.

Proof: Suppose in status *copying*, when x receives a *CPRly* from node g, it finds that there exists an attach-level, level-l, $0 \leq l \leq k$, where $k = |csuf(x.ID, g.ID)|$ for itself in $g.table$. (There must exist such a node g and level-l, since $x.ID$ is unique and at least $|V_{x[d-1]...x[0]}| < K$.) Then x changes status to *waiting*. So far, x has sent out at most $l+1$ *CP* (one for requesting a level of neighbor, from level-0 to level-l). In status *waiting*, x sends *JW* to u_1, u_2, and so on until a node u_j sends a positive reply to x. For each $u_{j'}$, $2 \leq j' \leq j$, $d-1 \geq |csuf(x.ID, u_{j'}.ID)| \geq |csuf(x.ID, u_{j'-1}.ID) \geq l$. Hence, x can at most send $d-i$ *JW* in status *waiting*. Therefore, the total number of *CP* and *JW* sent by x is at most $d+1$. ■

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 $\sum_{i=0}^{d-1} \frac{n}{b^i} P_i(n) - 1$, where $P_0(n)$ is $\sum_{j=0}^{K-1} \frac{C(b^{d-1}-1, j) C(b^d - b^{d-1}, n-j)}{C(b^d-1, n)}$; $P_i(n)$, for $1 \leq i < d-1$, is $\sum_{j=0}^{K-1} C(b^{d-1-i}-1, j) \sum_{k=K-j}^{\min(n-j, B)} \frac{C(B,k) C(b^d - b^{d-i}, n-k-j)}{C(b^d-1, n)}$, where $B = (b-1)b^{d-i-1}$; and $P_{d-1}(n)$ is $1 - \sum_{j=0}^{d-2} P_j(n)$.*

Proof: Suppose $V_x^{Notify} = V_\omega$. Then x needs to notify all the nodes in V_ω. By Proposition B.6, 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 $Z = |V_\omega|$. We denote the probability that Y equals j given $|V| = n$ as $P_j(n)$, $j \in [d]$. Then, $P_j(n) = P(|V_\omega| \geq K \wedge |V_{x[j]\cdot\omega}| < K)$, i.e., $P_j(n) = P(|V_{x[j-1]...x[0]}| \geq K \wedge |V_{x[j]...x[0]}| < K)$. Hence, we have

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

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

Next, we compute $P_i(n)$, $i \in [d-1]$. In general, IDs of nodes in V can be drawn from $b^d - 1$ possible values. That is, for any y, $y \in V$, $y.ID$ could be any value from 0 to $b^d - 1$ except $x.ID$.

If $i = 0$, then $|V_{x[0]}| < K$, 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, $P_0(n) = \sum_{j=0}^{K-1} \frac{C(b^{d-1}-1, j) C(b^d - b^{d-1}, n-j)}{C(b^d-1, n)}$

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 $x[i-1]...x[0]$). Hence, for $1 \leq i < d-1$, $P_i(n) = \sum_{j=0}^{K-1} C(b^{d-1-i}-1, j) \sum_{k=K-j}^{\min(n,B)} \frac{C(B,k) C(b^d - b^{d-i}, n-k-j)}{C(b^d-1, n)}$, where $B = (b-1)b^{d-i-1}$.

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

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

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

Theorem 6 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, 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 $\sum_{i=0}^{d-1} (\frac{n+m}{b^i}) P_i(n)$, where $P_i(n)$ is defined in Theorem 5.*

Proof: See the proof of **Theorem 5** in [6]. ■

Corollary B.8 *Suppose a set of nodes, $W = \{x_1,...,x_m\}$, $m \geq 2$, join a K-consistent network $\langle V, \mathcal{N}(V)\rangle$. Then for any node x, $x \in W$, an upper bound of the expected number of messages in the form of SN(x, y) or SNRly(x, y) sent by x, $y \neq x$, is $\sum_{i=0}^{d-1} (\frac{m}{b^{i+1}}(d-i-1)) P_i(n)$, where $n = |V|$ and $P_i(n)$ is defined in Theorem 5.*

Proof: See the proof of **Corollary A.6** in [6]. ■

The upper bound defined in Theorem 6 is also an upper bound for the expected number of messages in the form of *InSysNotiMsg*. See Section A.2 in [6] for arguments. Lastly, The number of messages in the form of *RN* and *RNRly* is $O(Kdb)$, 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* are piggyback'ed with some other messages, such as *JWRly* and *JNRly*. Hence, the number of messages in the form of *RN* and *RNRly* that is related to a joining node is at most $2Kdb$.