Building Self-Stabilizing Overlay Networks with the Transitive Closure Framework[☆]

Andrew Berns, Sukumar Ghosh[1], Sriram V. Pemmaraju[2]

Department of Computer Science
The University of Iowa
14 MacLean Hall
Iowa City, Iowa, USA 52242

Abstract

Overlay networks are expected to operate in hostile environments where node and link failures are commonplace. One way to make overlay networks robust is to design self-stabilizing overlay networks, i.e., overlay networks that can handle node and link failures without any external supervision. In this paper, we first describe a simple framework, which we call the *Transitive Closure Framework* (TCF), for the self-stabilizing construction of an extensive class of overlay networks. Like previous self-stabilizing algorithms for overlay networks, TCF permits intermediate node degrees to grow to $\Omega(n)$, independent of the maximum degree of the target overlay network. However, TCF has several advantages over previous work in this area: (i) it is a "framework" and can be used for the construction of a variety of overlay networks (e.g. Linear, Skip+), not just a particular network, (ii) it runs in an optimal number of rounds for a variety of overlay networks, and (iii) it can easily be composed with other non-self-stabilizing protocols that can recover from specific bad initial states in a memory-efficient fashion. We demonstrate the power of our framework by deriving from TCF a simple self-stabilizing protocol for constructing Skip+ graphs (Jacob et al., PODC 2009) that guarantees optimal convergence time from any configuration.

Keywords: self-stabilization, overlay networks, fault-tolerant algorithms, distributed algorithms, stabilization bounds

[☆] An early version of this paper appeared at SSS 2011.

Email addresses: andrew-berns@uiowa.edu (Andrew Berns), sukumar-ghosh@uiowa.edu (Sukumar Ghosh), sriram-pemmaraju@uiowa.edu (Sriram V. Pemmaraju)

[1] This work is supported in part by National Science Foundation grant CNS-0956780

[2] This work is supported in part by National Science Foundation grant CCF 0915543

1. Introduction

An *overlay network* is induced by logical or virtual links constructed over one or more underlying physical links. The use of virtual links enables designers to build any topology regardless of the underlying physical network, allowing the creation of networks with desirable properties, such as low diameter and mean path length (for efficient routing), low degree (for low memory requirements and maintenance overhead), low congestion, etc. Fault tolerance in overlay networks is an important focus for researchers and practitioners alike. Since nodes and links do not typically exist in stable and controlled environments, overlay networks must be prepared to handle unexpected node and link failures.

Traditionally, overlay networks are classified into two categories: structured and unstructured. Unstructured networks have no topological restrictions other than connectedness, and they are relatively simple to manage. Conversely, structured networks have "hard" topological constraints and recovery from bad configurations is a key challenge for such networks. Such bad configurations may be caused by node or link failures, by a node JOIN or a node LEAVE, or by deliberate actions of nodes trying to derive undue performance benefits for themselves.

One method for handling the hostile operating environments of overlay networks is *self-stabilization*[1]. Informally, a self-stabilizing overlay network is one that can build the "correct" configuration when starting from any weakly-connected initial configuration. Recently the design of self-stabilizing overlay networks has received attention – examples include algorithms for constructing *double-headed radix trees* [2], the LINEAR network [3], SKIP+ graphs [4], and CHORD-like networks [5]. The goal in these papers is to design self-stabilizing algorithms for overlay network construction; these algorithms run on the individual nodes of a weakly-connected network and, by node actions that include edge-additions and edge-deletions, the network is transformed into a legal overlay network. In a sense, these algorithms are taking a walk through the space of all networks defined by a given set of nodes, starting from a source network that is illegal and ending up at a target network that is legal. This algorithmic process is conceptually no different from, for e.g., starting with a (possibly unbalanced) binary search tree and transforming it into a *balanced* binary search tree (e.g., AVL tree) via a series of local rotations. However, since we are in a distributed setting, it is important that the illegality of a network be detected using only local information and fixed using local actions.

It is worth reemphasizing at the outset that since the edges of an overlay network are virtual and, in a sense, independent of the underlying physical links, these edges can be deleted and inserted as a result of program actions, or as a result of events such as JOINs and LEAVEs. This aspect of our model — the fact that the network may change repeatedly as the result of algorithm actions — makes it fundamentally different from other standard models of distributed computation such as $\mathcal{LOCAL}$ and $\mathcal{CONGEST}$ [6]. For example, it is easy to see that $\Omega(\text{DIAM}(G))$ is a lower bound on the problem of constructing a minimum spanning tree (MST) of a network G in a distributed setting in the $\mathcal{LOCAL}$ model. However, this lower bound is mainly due to the fact that the underlying

network G has to remain static – once we allow program actions to modify the underlying network, this lower bound no longer holds and using techniques described in this paper, it is easy to see that $O(\log n)$ rounds suffice for MST construction, independent of $\textsc{Diam}(G)$.

1.1. Our Contributions

In this paper, we first describe a simple framework, which we call the *Transitive Closure Framework* (TCF), for the self-stabilizing construction of an extensive class of overlay networks. This is a "framework" rather than an algorithm and by instantiating certain subroutines in this framework we can obtain self-stabilizing algorithms for specific overlay networks. Like previous self-stabilizing overlay networks, TCF permits node degrees to grow to $\Omega(n)$, independent of the maximum degree of the target overlay network. However, TCF has several advantages over previous work in this area: (i) it is a "framework" and can be used for the construction of a variety of overlay networks, not just a particular network, (ii) it runs in optimal number of rounds for a variety of overlay networks, and (iii) it can easily be composed with other non-self-stabilizing protocols that can recover from specific bad initial states in a memory-efficient fashion. We elaborate on items (ii) and (iii) below.

- We identify a natural parameter of overlay networks, namely the *detector diameter*. Consider a set of nodes V, a legal overlay network $G = (V, E)$ on V, and a faulty network $G_f = (V, E_f)$ on V. Typically, the diameter of G, denoted $\textsc{Diam}(G)$, is small relative to $|V|$, whereas $\textsc{Diam}(G_f)$ may be much larger than $\textsc{Diam}(G)$. Now let $V' \subseteq V$ denote a set of node that are "detectors," i.e., nodes in G_f whose local states alert them to the fact that the overlay network is faulty (definitions of these notions appear in Sect. 2.1). Assume for now that V' is non-empty. How detectors are distributed throughout G_f is crucial to G_f's stabilization time. If every node is within some distance d of a detector in G_f, then any algorithm obtained from TCF can transform G_f to G in $O(d + \log n)$ rounds, where $n = |V|$. To place this in context, we then show a lower bound: the worst-case self-stabilization time of a silent algorithm is bounded below by $\Omega(\textsc{Diam}(G))$. The natural question to ask is: what is the gap between the lower bound $\Omega(\textsc{Diam}(G))$ and the upper bound $O(d + \log n)$ for any G_f? It may seem as though d can be as large as $\textsc{Diam}(G_f)$), which, as we mentioned earlier, can be much larger than $\textsc{Diam}(G)$. In one of our main results, we show that d is no more than $\textsc{Diam}(G) + 1$ for Skip+ and Linear networks, thus showing that the self-stabilization time of algorithms derived from TCF is within an additive logarithmic-factor of the optimal time for these networks.

- The above discussion of the self-stabilization time of TCF ignores the maximum node degree increase allowed during recovery by TCF. As mentioned earlier, TCF requires all node degrees to become $\Theta(n)$ during the algorithm execution before the degrees drop down to their final values in

the target overlay network G. However, to offset this memory-inefficiency we show that TCF can be easily composed with other, (possibly non-self-stabilizing) overlay network protocols that can deal with specific initial states in a memory-efficient manner. We introduce the Local Repair Framework (LRF), which allows for the efficient repair of certain failures. To demonstrate this, we create a JOIN protocol for SKIP+ graphs that (i) stabilizes from *arbitrary* initial configurations in $O(\log n)$ rounds, while permitting an $O(n)$ degree increase and (ii) stabilizes from a single-JOIN state in $O(\log n)$ rounds, while degrees of non-joining nodes remain bounded by $O(\log n)$ during stabilization.

Finally, we demonstrate the power of our framework by deriving from TCF, a simple self-stabilizing protocol for constructing SKIP+ graphs [4]. The SKIP+ graph is a locally-checkable extension of SKIP graphs [7]. We show that the detector diameter for an n-node SKIP+ graph is $O(\log n)$ and therefore our algorithm runs in $O(\log n)$ rounds, exactly matching the lower bound of $\Omega(\log n)$, which follows from the well-known fact that the diameter of an n-node SKIP+ graph is $\Theta(\log n)$. Since a single-node JOIN operation can be performed in $O(\log n)$ rounds by performing the composition alluded to above, we obtain a self-stabilizing overlay network that (i) stabilizes from arbitrary initial configurations in $O(\log n)$ rounds (which is optimal), while permitting an $O(n)$ degree increase and (ii) stabilizes from a single-JOIN state in $O(\log n)$ rounds, while permitting only an $O(1)$ degree increase.

1.2. Paper Organization

Our paper is organized as follows. In Section 2, we describe our model and formally define the problem. Section 3 describes the Transitive Closure Framework and its performance, while Section 4 describes an implementation of the Transitive Closure Framework for SKIP+ graphs. We describe a way to incorporate non-self-stabilizing components into our framework in Section 5, and provide an example of JOIN in SKIP+ graphs in Section 6. Finally, we provide several concluding thoughts in Section 7.

2. Preliminaries

2.1. Model and Definitions

Let V be a set of nodes. We suppose that there are two functions $\texttt{id} : V \to \mathbb{Z}^+$ and $\texttt{rs} : V \to \{0,1\}^*$ that associate with each node in V a unique identifier and a random bit string. Where clear from context, we shall refer to a node simply by its identifier. The association of `id`-values to nodes is adversarial, however it is assumed that the adversary assigns `id`-values to nodes without having access to their `rs`-values. A *family of overlay networks* is defined as a mapping $ON : \Lambda \to \mathcal{G}$, where Λ is the set of all triples $\lambda = (V, \texttt{id}, \texttt{rs})$ and $\mathcal{G}$ is the set of all labeled undirected graphs. The mapping ON satisfies the property that for any $\lambda = (V, \texttt{id}, \texttt{rs})$, ON maps λ to a graph with vertex set

V and whose vertex labels are assigned by the functions $\texttt{id} : V \to \mathbb{Z}^+$ and $\texttt{rs} : V \to \{0,1\}^*$. At this point, the only other assumption we make about ON is that it is well-defined for every member $\lambda \in \Lambda$.

Let E be an arbitrary set of undirected edges on V such that the graph $G = (V, E)$ is connected. Our goal is to design an algorithm that starts on any given labeled graph $(G = (V, E), \texttt{id}, \texttt{rs})$ and computes the overlay network $ON(\lambda)$, where $\lambda = (V, \texttt{id}, \texttt{rs})$. Such an algorithm is a *self-stabilizing algorithm* for computing the family of overlay networks ON. We now explain what it means precisely for an algorithm to compute $ON(\lambda)$. Each node $v \in V$ maintains, over the course of the algorithm, a set of neighbors $N(v)$, as part of its local state. $N(v)$ is node v's view of the network that it is part of. More precisely, the "current" network consists of the edges $\{(v, w) | w \in N(v)\}$ for all $v \in V$. Initially, the sets $N(v)$, for all nodes $v \in V$ induce E (the input set of edges). We use $S(v)$ to denote the local state of a node $v \in V$. $S(v)$ consists of $N(v)$ plus additional local variables that presumably help node v in its computations. We assume a synchronous message-passing model. In each synchronous round, node v reads the messages it received in the previous round from its neighbors, updates $N(v)$ if necessary, and sends messages to its neighbors in $N(v)$. Message sizes are assumed to be unbounded and typically in each round the messages sent by v consist of the $\texttt{id}$-values and $\texttt{rs}$-values of all the nodes $N(v)$. We assume reliable communication – all messages received were sent by a node, and all sent messages are received. Note that throughout the algorithm, node v can communicate with all of its current neighbors, i.e., nodes in $N(v)$ in one round of communication. Since $N(v)$ is continuously changing, which nodes v is able to communicate with in one round is also continuously changing as the algorithm executes. As mentioned before, this aspect of our model makes it fundamentally different from other standard models of distributed computation such as $\mathcal{LOCAL}$ and $\mathcal{CONGEST}$ [6].

To help make exposition easier, we followed the convention used in prior research and assumed undirected edges. Notice, however, that our results can easily be converted to the directed edge setting. In our model, any directed edge (u, v) can be converted into a bi-directional edge by having node u send a message to node v with u's information, and in one round node v can then add u to it's neighborhood.

Let $N_{ON(\lambda)}(v)$ denote the set of neighbors of node v in overlay network $ON(\lambda)$. A node $v \in V$ is said to be *faulty* in a particular round if its current set of neighbors $N(v)$ is not equal to $N_{ON(\lambda)}(v)$. The network is said to be *faulty* in a particular round if some node in V is faulty. The goal of our algorithm is to lead the network into a non-faulty state. Note that a faulty node v may not know that it is faulty, as node v's knowledge of the network is limited, and v's limited knowledge may suggest the network is correct. More precisely, let V' denote the current 2-neighborhood of v, let $\texttt{id}'$ and $\texttt{rs}'$ be the restrictions of $\texttt{id}$ and $\texttt{rs}$ respectively to V', and let λ' denote the triple $(V', \texttt{id}', \texttt{rs}')$. Node v may be able compute its "local" overlay network $ON(\lambda')$ and this may be consistent with the edges that v sees in its 2-neighborhood. In such a case, node v has no reason to believe that it is faulty, though it may be faulty because of other

nodes in V that are outside its 2-neighborhood. We say that an overlay network family ON is *locally checkable* if (i) every faulty configuration contains at least one node that detects a fault in its 2-neighborhood, and (ii) no node detects a fault in the correct configuration. As a "toy" example of a locally checkable overlay network family, consider the LINEAR network that consists of a path of nodes arranged in the order of node identifiers. For the LINEAR network, bad configurations include those in which a node perceives two neighbors both with smaller (or larger) identifiers, or those in which a node has three or more neighbors. It is easy to see LINEAR is locally checkable.

In the current model of computation, two efficiency metrics make most sense [3]. The first is the traditional worst-case number of synchronous *rounds* needed to terminate or stabilize (i.e., reach the target overlay network), and the second is the the maximum *degree increase* of a node during algorithm execution. This second measure may be viewed as the amount of "extra memory" that nodes consume during algorithm execution. Another view of this measure is as follows. Typically, overlay networks have small maximum degree (relative to number of nodes in the network) and if we start off with an illegal overlay network in which all degrees are small, then the requirement that the maximum node degree increase be bounded forces the algorithms we construct to travel through the space of only low-degree networks before reaching its destination overlay network. Ideally, from the point of view of scalability, both measures should be sublinear, preferably polylogarithmic, in the number of nodes currently in the network. However, no existing algorithms seem to have achieved this for non-trivial overlay networks. For example, Jacob et al. [4] present a self-stabilizing algorithm for building a SKIP+ graph in $O(\log^2 n)$ rounds (with high probability). However, the worst-case memory requirements of this algorithm are linear, and the algorithm and its proof of correctness are both quite complex.

2.2. SKIP+ Graphs

In this paper, we use SKIP+ graphs to illustrate the utility of TCF. SKIP graphs were introduced in 2003 by Aspnes and Shah [7] as a fault-tolerant distributed data structure for efficient searching in peer-to-peer systems. In a SKIP graph, each node u has a unique identifier $u.id$, as well as a random sequence, $u.rs$. We assume the random sequence length is fixed for the given set of nodes. In many settings, it is assumed the random sequence length is sufficient enough that no node has the same random sequence as any other node. For our purposes, then, we will assume the random sequence is of length $c \cdot \log n$, for some constant c. To help with defining SKIP graphs and SKIP+ graphs, we provide the following notation, taken with slight modification from [4].

- $pre_i(u)$: for any node u and nonnegative integer i, $pre_i(u)$ denotes the leftmost (most significant) i bits of $u.rs$
- $pred(u, W)$: for any node u and subset W of nodes, $pred(u, W)$ is the

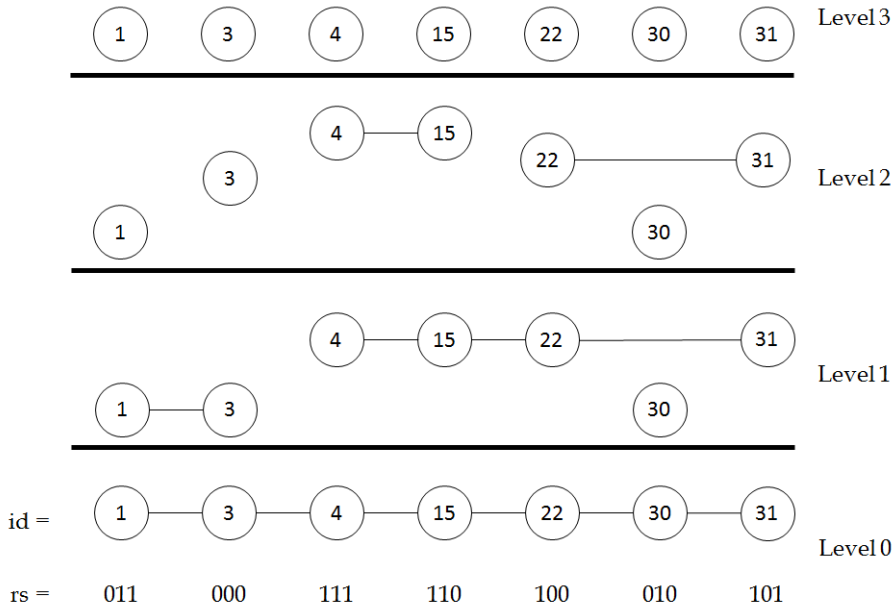

Figure 1: An illegal Skip graph in which none of the nodes are able to detect a fault (an edge should exist between nodes 3 and 30).

node in the set W with largest id whose id is less than $u.id$. If no such node exists, $pred(u, W) = \perp$.

- $succ(u, W)$: for any node u and subset W of nodes, $succ(u, W)$ is the node in the set W with smallest id whose id is more than $u.id$. If no such node exists, $pred(u, W) = \perp$.

A legal Skip graph consists of levels $0, 1, 2, \ldots$. At each level i, a node u has at most two neighbors: $pred(u, \{w | pre_i(w) = pre_i(u)\})$ and $succ(u, \{w | pre_i(w) = pre_i(u)\})$, assuming such nodes exist. It is shown in [7] that with high probability (w.h.p.), the degree of each node in a Skip graph is $O(\log n)$ and furthermore there is a simple protocol that can search for a node in $O(\log n)$ rounds. This result depends upon the adversary assigning identifiers to nodes being oblivious to the random sequences.

While these are highly desirable properties, a problem for designing self-stabilizing Skip graphs is that Skip graphs are not locally checkable. To see this consider Fig. 1. Here, each node believes the topology is correct, when in fact this is not a legal Skip graph, as there should be an edge between node 3 and node 30 in Level 1. However, these two nodes are unaware of the existence of each other and other nodes in the vicinity are unable to help.

To create a locally-checkable version of the Skip graph, additional links are needed and this leads us to the definition of Skip+ graphs [4]. In a Skip+

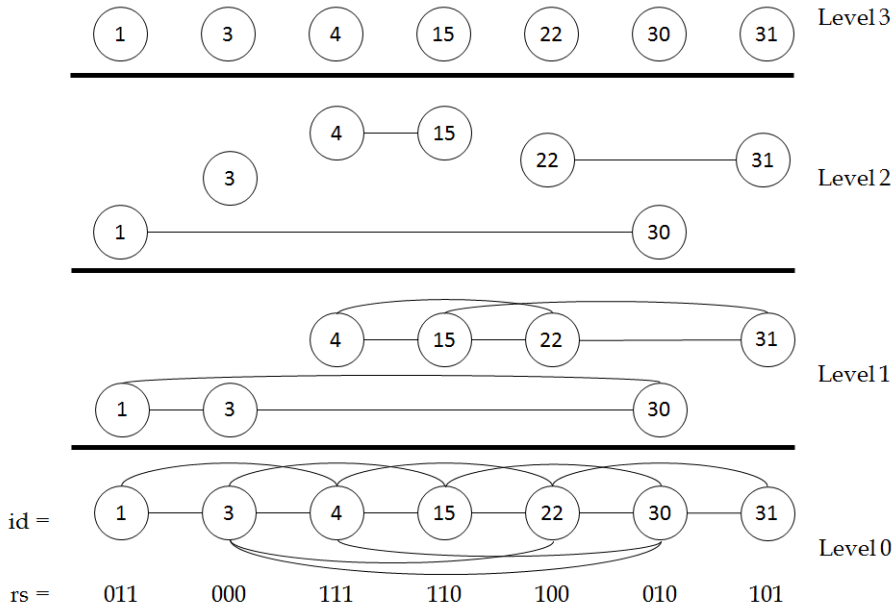

Figure 2: A Skip+ graph. Notice the edges of the Skip+ graph are a superset of the edges for the corresponding Skip graph. The additional edges ensure the graph is locally checkable.

graph, as in a Skip graph, each node u has a unique identifier $u.id$, as well as a random sequence $u.rs$. Below we present additional definitions that we need for defining Skip+ graphs.

- For node u, nonnegative integer i, and $x \in \{0, 1\}$,

$$pred_i(u, x) = pred(u, \{w \mid pre_{i+1}(w) = pre_i(u) \cdot x\})$$

  In words, $pred_i(u, x)$ is the predecessor for u, selected from all nodes who have the same length-i prefix as u and whose $(i + 1)$ bit is x.

- For node u, nonnegative integer i, and $x \in \{0, 1\}$,

$$succ_i(u, x) = succ(u, \{w \mid pre_{i+1}(w) = pre_i(u) \cdot x\})$$

- For node u and nonnegative integer i,

$$low_i(u) = \min\{pred_i(u, 0).id, pred_i(u, 1).id\}$$

- For node u and nonnegative integer i,

$$high_i(u) = \max\{succ_i(u, 0).id, succ_i(u, 1).id\}$$

- For node u and nonnegative integer i,

$$range_i(u) = [low_i(u), high_i(u)]$$

In a legal SKIP+ graph, a node u has edges to all nodes v such that $v.id \in range_i(u)$ and $pre_i(u) = pre_i(v)$. These nodes v are referred to as the *level-i neighbors* of u. A legal SKIP+ graph is shown in Fig. 2.

3. Transitive Closure Framework

The *Transitive Closure Framework* (TCF) is shown in Program 1. TCF uses a predicate called `DETECT` and a subroutine called `REPAIR` – these need to be instantiated appropriately for specific families of overlay networks. To describe TCF we first introduce some notation and definitions.

Let $\lambda = (V, \mathtt{id}, \mathtt{rs})$ be a labeled set of nodes. Let $G = (V, E)$ be an arbitrary undirected graph on the set of nodes V. Recall that $N(v)$ is the set of neighbors of v, i.e., $\{w \mid (v, w) \in E\}$. For each node v let $\mathtt{id}_v$ and $\mathtt{rs}_v$ respectively be the restrictions of `id` and `rs` to $N(v)$. Let $N^2(v)$ denote the set of "at most 2-hop neighbors" of v, i.e., $\{v\} \cup N(v) \cup \{w \mid (u, w) \in E \text{ and } u \in N(v)\}$. Let $\mathtt{id}_v^2$ and $\mathtt{rs}_v^2$ be the functions `id` and `rs` restricted to the set $N^2(v)$. Then $G_v^2 := ON(N^2(v), \mathtt{id}_v^2, \mathtt{rs}_v^2)$ is the legal overlay network on $N^2(v)$. For node v, based on its local information, this is the "local" version of the overlay network it wants to see for the network to be in a legal configuration. Let E_v be the set of all edges that node v is aware of, i.e., $E_v = \{(v, u) | u \in N(v)\} \cup \{(u, w) | u \in N(v)\}$.

Definition 1. *The `DETECT` predicate at a node v, evaluated over $(N^2(v), \mathtt{id}_v^2, \mathtt{rs}_v^2), E_v$, is true exactly when $N(w) = N_{ON(N^2(v), \mathtt{id}_v, \mathtt{rs}_v)}(w)$ for all $w \in N(v) \cup \{v\}$; otherwise the `DETECT` predicate is false. A node v is called a* detector *if the `DETECT` predicate evaluates to true at node v.*

For an example of a `DETECT` predicate, consider the LINEAR family of overlay networks again [3]. Here, for a given $\lambda = (V, id, rs)$, LINEAR(λ) is a path $(v_1, v_2, \ldots, v_n)$ formed by nodes in V such that $v_1.id < v_2.id < \cdots < v_n.id$. In this case, the `DETECT` predicate at a node v is true iff the 2-neighborhood of v does not induce a path (with at most 5 nodes) that is sorted by `id`s.

Definition 2. *Given $\lambda = (V, \mathtt{id}, \mathtt{rs})$, the `REPAIR` subroutine at node v sets the neighborhood of v to $N_{ON(\lambda)}(v)$ in one round.*

Each node u has an associated boolean variable $detect_u$ that becomes true in one of two ways: (i) when the predicate `DETECT` is true for u (Line 4) and (ii) when $detect_v$ is true for some neighbor $v \in N(u)$ in a previous round (Line 12). $detect_u$ becoming true in the second manner causes the spreading of the "$detect = true$" event through the network. As this event spreads through the network, all nodes for which the boolean variable *detect* is true participate in the transitive closure process (Line 11). In other words, each node with $detect = true$ in its neighborhood, expands its neighborhood to include all the

neighbors of its neighbors. Soon enough, every node has $detect = true$ and the network becomes a clique. At this point (which is detected in Line 7), the ideal neighborhood of each node can be built using the `REPAIR()` subroutine (Line 8).

Program 1 Transitive Closure Framework

Program for process u
Variables: neighborhood $N(u)$, Boolean $detect_u$

in each round do
1. Send $N(u)$, $\mathtt{id}_u$, and $\mathtt{rs}_u$ to every neighbor $v \in N(u)$
2. Receive $N(v)$, $\mathtt{id}_v$, and $\mathtt{rs}_v$ from each $v \in N(u)$
3. Compute from this information: $\lambda^2 \leftarrow (N^2(u), \mathtt{id}_u^2, \mathtt{rs}_u^2)$ and
$$E_u \leftarrow \{(u,v)|v \in N(u)\} \cup \{(v,w)|v \in N(u)\}$$
4. $detect_u \leftarrow \mathtt{DETECT}(\lambda^2, E_u) \vee\ detect_u$
5. Send $detect_u$ to every neighbor $v \in N(u)$
6. Receive $detect_v$ from every neighbor $v \in N(u)$
7. **if** $detect_u \wedge \forall v \in N(u) : (detect_v \wedge (\{N(v) \cup \{v\}\} = \{N(u) \cup \{u\}\}))$ **then**
8. $\quad N(u) \leftarrow \mathtt{REPAIR}(N(u) \cup \{u\}, \mathtt{id}_u, \mathtt{rs}_u)$
9. $\quad detect_u \leftarrow false$
10. **else if** $detect_u \vee (\bigvee_{v \in N(u)} detect_v)$ **then**
11. $\quad N(u) \leftarrow N(u) \cup \{\bigcup_{v \in N(u)} N(v)\}$ //transitive closure
12. $\quad detect_u \leftarrow true$
13. **fi**

od

Because algorithms derived from the TCF are initiated by detectors, the efficiency of these algorithms depends, to a significant extent, on the distribution of detectors in the network. For overlay networks that are not locally checkable (e.g., Skip graphs) the presence of even a single detector is not guaranteed when the network is faulty – this is in fact the primary motivation of Jacob et al. [4] for defining Skip+ graphs. To formalize the notion of the distribution of detectors, we define the *detector diameter* $\textsc{DetDiam}_{ON}(n)$ of any family ON of overlay networks as follows.

Fix a family of overlay networks ON. Let $\lambda = (V, \mathtt{id}, \mathtt{rs})$ be a labeled set of nodes and let E be an arbitrary set of undirected edges on V such that $G = (V, E)$ is connected. Think of G as representing a possibly faulty overlay network and let $D \subseteq V$ be the set of detectors in G. The nodes in D are independent of any specific algorithm – whether or not a node is a detector depends solely upon the ideal network configuration (i.e., $ON(\lambda)$) and the current network configuration (i.e., G). Note that D may be empty, either because $G = ON(\lambda)$ or because even if $G \neq ON(\lambda)$, no node is able to detect this. If we assume that ON is a family of locally checkable overlay networks then $G \neq ON(\lambda)$ implies that $D \neq \emptyset$. The *detector diameter of G with respect to λ*, denoted $\textsc{DetDiam}_{ON(\lambda)}(G)$, is the maximum hop distance in G

between any node in V and the closest detector. The implication of this definition is that if the initial state of the system is network G, then some node v in G is $\textsc{DetDiam}_{ON(\lambda)}(G)$ hops from the closest detector and thus the TCF algorithm initiated by detectors requires $\textsc{DetDiam}_{ON(\lambda)}(G)$ rounds to reach v. The *detector diameter* $\textsc{DetDiam}_{ON}(n)$ of a family ON of overlay networks is the maximum of $\textsc{DetDiam}_{\lambda}(G)$ over all $\lambda = (V, id, rs)$ with $|V| = n$ and all connected networks $G = (V, E)$. It is worth noting that if random strings are indeed used to define the family ON of overlay networks, then λ, D_λ, and $\textsc{DetDiam}_{ON}(n)$ are all random variables.

We are able to show the following upper bound on the self-stabilization time of TCF.

Theorem 1. *The Transitive Closure Framework presented in Program 1 is a self-stabilizing algorithm for constructing any family of overlay networks ON in at most $\textsc{DetDiam}_{ON}(n) + \log(n) + 1$ rounds.*

Proof. Fix a family ON of overlay networks and consider an arbitrary faulty network G. Since ON is locally-checkable, there is at least one node u in G that is a detector. Node u will set $detect_u$ to true (line 4), and all neighbors $v \in N(u)$ will set $detect_v$ to true in the following round (lines 10-12). In general, a node x at distance i from a detector will set $detect_x$ to true in at most i rounds. Therefore, in at most $\textsc{DetDiam}_{ON}(n)$ rounds, all nodes are aware of a faulty configuration and are executing the transitive closure process (lines 11-12 of 1).

Nodes executing the transitive closure process will continue to do so until the network is completely connected (line 7). When all nodes are executing the transitive closure process, the diameter of the network is halved in every round, since in every round each node becomes neighbors with all nodes that were distance 2 from it. Therefore, in at most $\log n$ rounds, the network is completely connected. At this point, the condition on line 7 is true, and all nodes execute the `REPAIR(`$N(u) \cup u$`)` action in the same round. The correct topology is built, and no more detectors are present in the network.

Therefore, the Transitive Closure Framework in Program 1 is self stabilizing, and requires at most $\textsc{DetDiam}_{ON}(n) + \log n + 1$ rounds to build the correct overlay network ON. □

3.1. A Lower Bound

This section is devoted to the proof of the following lower bound on the stabilization time for constructing locally checkable overlay networks. Recall that a self-stabilizing algorithm is *silent* if communication eventually ceases and no actions remain enabled. Let $\textsc{Diam}_{ON}(n)$ be the maximum diameter of any at-most-n-node member of ON.

Informally, we show the lower bound of stabilization time as follows. Let $ON(\lambda)$ be the correct overlay network for some $\lambda = (V, \texttt{id}, \texttt{rs})$, and let the diameter of $ON(\lambda)$ be d. Label the nodes on the path realizing the diameter as $p_0, p_1, \cdots, p_d$. We compare the diameter of $ON(\lambda)$ d with the distance between nodes p_1 and p_d in $G'ON(\lambda')$, where $\lambda' = (V \setminus \{p_0\}, \texttt{id}, \texttt{rs})$ (that is, λ' is the

set of nodes with node p_0 removed). If p_1 and p_d are "far apart" (relative to d) in G', we show connecting p_0 to p_d in G' requires $\Omega(d)$ time to stabilize. If p_1 and p_d are "close" (relative to d) in G', we show the removal of p_0 from $ON(\lambda)$ creates a faulty node at distance $\Omega(d)$ from any detector, and therefore $\Omega(d)$ rounds are again required to stabilize. We provide the formal lower bound and proof below.

Theorem 2. *Let ON denote any family of locally checkable overlay networks. Any silent self-stabilizing algorithm for constructing ON takes, in the worst case, $\Omega(\textsc{Diam}_{ON}(n))$ time.*

Proof. Let $\lambda = (V, \texttt{id}, \texttt{rs})$ with $|V| = n$. Let $G = ON(\lambda)$ and suppose that $d = \textsc{Diam}_{ON}(G)$. There exists a shortest path consisting of distinct nodes $p_0, p_1, \cdots, p_d$ in the network G. Let $V' = V \backslash \{p_0\}$ and $\texttt{id}'$ and $\texttt{rs}'$ be restrictions of $\texttt{id}$ and $\texttt{rs}$ respectively to V'. Let $\lambda' = (V', \texttt{id}', \texttt{rs}')$ and $G' = ON(\lambda')$ (such a G' must exist, as we have assumed $ON : \Lambda \to \mathcal{G}$ is defined on all members of Λ). Note that G' is the correct ON network with node p_0 removed. We consider the distance between nodes p_1 and p_d in G' to complete our proof.

Case 1: $dist_{G'}(p_1, p_d) > \frac{d}{2}$. Insert node p_0 as a neighbor to node p_d in G'. The modified network is now faulty, as p_0 must be a neighbor of p_1 (and vice-versa) in the ideal configuration $ON(\lambda)$. Furthermore, only p_d and its immediate neighbors have knowledge of node p_0, and these nodes are at least $\frac{d}{2} + 1$ away from node p_1, a node that needs to change its local state. Information that the network is faulty can travel at most one hop per round, and no node on the path from p_1 to p_d is a detector before receiving this information. Therefore, the self-stabilization time from such a state is at least $\frac{d}{2} + 1$ (the time required for p_1 to be aware of p_0).

Case 2: $dist_{G'}(p_1, p_d) \leq \frac{d}{2}$. Let $B(p_d, \frac{d}{2})$ be the subgraph induced by all nodes at distance at most $\frac{d}{2}$ from node p_d in G, and let $B'(p_d, \frac{d}{2})$ be the subgraph induced by all nodes at distance at most $\frac{d}{2}$ from node p_d in G'. Notice that $B(p_d, \frac{d}{2})$ does *not* include node p_1, while $B'(p_d, \frac{d}{2})$ *does* include p_1. There must exist at least one node $w \in B(p_d, \frac{d}{2})$ whose neighborhood in G' is different from its neighborhood in G. If this were not the case, then $B(p_d, \frac{d}{2}) = B'(p_d, \frac{d}{2})$, which would be a contradiction. Notice that node w is at least distance $\frac{d}{2}$ from p_0 in G, and that the removal of p_0 must cause w to change its neighborhood. However, only immediate neighbors of p_0 are aware of its removal, and information that the network is faulty can travel at most one hop per round. Therefore, it will require at least $\frac{d}{2}$ rounds before node w is aware it should begin changing its neighborhood. Self-stabilization therefore requires at least $\frac{d}{2}$ rounds in this case.

□

3.2. Bounding the Detector Diameter

Theorem 1 yields an $O(\textsc{DetDiam}_{ON}(n) + \log n)$ upper bound on the self-stabilization time of TCF, whereas Theorem 2 yields an $\Omega(\textsc{Diam}(ON(n)))$ lower

bound on the running time of any silent self-stabilizing algorithm. Clearly, then, for overlay networks where $\textsc{Diam}(ON(n)) = \Omega(\textsc{DetDiam}_{ON}(n))$, the upper and lower bounds for overlay network construction are within an additive $O(\log n)$ of each other. Several overlay networks fall into this class. For instance, $\textsc{Diam}(\textsc{Linear}(n)) = n$, and clearly $\textsc{DetDiam}_{\textsc{Linear}}(n) \leq n$. Therefore, for Linear, our convergence bound is tight. We will show in Lemma 1 that $\textsc{Diam}(\textsc{Skip+}(n)) = \Omega(\textsc{DetDiam}_{\textsc{Skip+}}(n)$ as well. In fact, we suspect this holds for all non-trivial locally-checkable overlay networks, although we are unable to prove this at this point.

4. TCF for SKIP+ Graphs

To provide an example of how the Transitive Closure Framework can be used to create a specific topology, we consider the Skip+ graph [4]. As mentioned earlier, the Skip+ graph is a locally-checkable extension of the Skip graph. In the following section, we describe the Skip+ graph, and show how our TCF instantiates a self-stabilizing algorithm that creates a Skip+ graph within $O(\log n)$ of optimal time.

4.1. The DETECT Predicate and REPAIR Subroutine

For Skip+ graphs, both `DETECT` and `REPAIR` follow trivially from the definition of a Skip+ graph – that is, each node simply computes its range in the ideal Skip+ graph using its 2-neighborhood, and either creates links within this range (with the `REPAIR` subroutine) or compares this range with the current 2-neighborhood (with the `DETECT` predicate).

4.2. Analysis of the Transitive Closure Framework

To evaluate the performance of the TCF with regards to Skip+ graphs, we provide the detector diameter $\textsc{DetDiam}_{ON}(n)$.

Lemma 1. *The detector diameter for the family of* Skip+ *overlay networks* $\textsc{DetDiam}_{\textsc{Skip+}}(n) = c \cdot \log n + 1$.

Proof. Let G_ρ be a subgraph of an arbitrary network G, induced by the set of nodes $\{v : pre_{|\rho|}(v) = \rho\}$. Furthermore, let C_ρ be some connected component in G_ρ, V_{C_ρ} be the nodes from C_ρ, and $\mathtt{id}_{C_\rho}$ and $\mathtt{rs}_{C_\rho}$ be the identifiers and random sequences of nodes in V_{C_ρ}. Notice that the maximum diameter of $\textsc{Skip+}(V_{C_\rho}, \mathtt{id}_\rho, \mathtt{rs}_\rho)$ is $L - |\rho| + 1$, where $L = |u.rs|$ (each hop can bring a node at least one bit closer in random sequence to any other node). Recall that the random sequence length is $c \cdot \log n$. We will find the detector diameter by using induction on the length of ρ.

Base Case: $|\rho| = L$. In this case, $\forall v, v' \in C_\rho : v.rs = v'.rs$. Thus, either the nodes in C_ρ are completely connected (and thus $C_\rho = \textsc{Skip+}(V_{C_\rho}, \mathtt{id}_\rho, \mathtt{rs}_\rho)$), or some node $u \in C_\rho$ has a neighbor $v \in C_\rho$, and v has a neighbor $v' \in C_\rho$ such that v' is not a neighbor of u. In this case, node u (v) has v' (u) in its 2-neighborhood, and will detect that a direct link should exist but does not. Therefore, u and

v' will become detectors, and each node is at most $1 = (L - L) + 1$ hops away from a detector.

Let $\sigma \cdot x$ be a bit sequence consisting of σ concatenated with the bit x, and $\sigma \cdot \bar{x}$ be bit sequence made up of σ concatenated with the opposite of bit x.

Induction Hypothesis: Assume for all connected components in $G_{\sigma \cdot x}$ and $G_{\sigma \cdot \bar{x}}$, it holds that every node in a faulty connected component is within $L - (|\sigma| + 1) + 1 = L - |\sigma|$ steps of a detector.

Inductive Step: Consider the connected component $C_{\sigma \cdot x}$, which joins with 0 or more other connected components to form C_σ. We assume C_σ does not match the ideal SKIP+ graph corresponding to the nodes of C_σ, else our lemma is vacuously true. We examine the following cases.

Case 1: $C_{\sigma \cdot x} = C_\sigma$. If $C_{\sigma \cdot x} = C_\sigma$, then all nodes in C_σ are at most $L - |\sigma|$ from a detector by the induction hypothesis, and our claim holds.

Case 2: $C_{\sigma \cdot x} \neq \text{SKIP+}(V_{C_{\sigma \cdot x}}, \texttt{id}_{C_{\sigma \cdot x}}, \texttt{rs}_{C_{\sigma \cdot x}})$. If $C_{\sigma \cdot x}$ was a faulty configuration, our claim again holds by the induction hypothesis, as all nodes in $C_{\sigma \cdot x}$ remain within $L - |\sigma|$ hops of a detector in C_σ.

Case 3: $C_{\sigma \cdot x} \neq C_\sigma$ *and* $C_{\sigma \cdot x} = \text{SKIP+}(V_{C_{\sigma \cdot x}}, \texttt{id}_{C_{\sigma \cdot x}}, \texttt{rs}_{C_{\sigma \cdot x}})$. There must be some edge in C_σ that was not present in $C_{\sigma \cdot x}$, or else $C_\sigma = C_{\sigma \cdot x}$. Furthermore, notice these new edges may only connect nodes in $C_{\sigma \cdot x}$ to nodes from $G_{\sigma \cdot \bar{x}}$ (otherwise, C_σ would have been a single connected component in $G_{\sigma \cdot x}$).

First consider the case where $C_{\sigma \cdot x}$ connects with exactly one other connected component $C_{\sigma \cdot \bar{x}}$ to form C_σ. If there exists a node in $C_{\sigma \cdot x}$ that is not connected to a node in $C_{\sigma \cdot \bar{x}}$, then there must exist $v \in C_{\sigma \cdot x}$ such that v has no neighbor $x \in C_{\sigma \cdot \bar{x}}$ and v has a neighbor $w \in C_{\sigma \cdot x}$ such that w has a neighbor $x' \in C_{\sigma \cdot \bar{x}}$. Node v will then have x' inside it's calculated range at level $|\sigma|$, but no direct link to x', and v is a detector. By assumption, v is within $L - |\sigma|$ of all other nodes in $C_{\sigma \cdot x}$. If all nodes in $C_{\sigma \cdot x}$ *are* connected to at least one node in $C_{\sigma \cdot \bar{x}}$, then either there is a detector in $C_{\sigma \cdot \bar{x}}$ that (by the induction hypothesis) is within $L - |\sigma| + 1$ of all nodes in $C_{\sigma \cdot x}$, or $C_{\sigma \cdot \bar{x}}$ was correct, in which case the diameter of the network is at most $L - |\sigma| + 1$, and our claim holds.

Next, consider where $C_{\sigma \cdot x}$ is connected to more than one component. As before, if any node in $C_{\sigma \cdot x}$ has no edge to a node in $G_{\sigma \cdot \bar{x}}$, then there must exist a node $v \in C_{\sigma \cdot x}$ without an edge to any node in $G_{\sigma \cdot \bar{x}}$, and v must have a neighbor $w \in C_{\sigma \cdot x}$ such that v has a neighbor from $G_{\sigma \cdot \bar{x}}$. Therefore, v will be a detector, and our claim holds.

Consider where all nodes in $C_{\sigma \cdot x}$ are connected to at least one node in $G_{\sigma \cdot \bar{x}}$. If a node $v \in C_{\sigma \cdot x}$ has a link to nodes in two different connected components, say $C_{\sigma \cdot \bar{x}}$ and $C'_{\sigma \cdot \bar{x}}$, then v will detect that these nodes should be connected but are not, and v becomes a detector. If all nodes in $C_{\sigma \cdot x}$ are connected to at least one node from $G_{\sigma \cdot \bar{x}}$, and no node has connections to nodes from $C_{\sigma \cdot \bar{x}}$ and $C'_{\sigma \cdot \bar{x}}$, then there must exist two neighbors $v, w \in C_{\sigma \cdot x}$ such that v and w share no neighbors from $G_{\sigma \cdot \bar{x}}$. In this case, both v and w are detectors, and our claim holds.

To find the detector diameter, set ρ to the empty string. Then, every node is at most distance $L+1$ from a detector in a faulty configuration, and therefore $\text{DETDIAM}_{\text{SKIP+}}(n) = L + 1$. □

The Transitive Closure Framework allows us to provide the following theorem, which is easily derived using Theorem 1.

Theorem 3. *The Transitive Closure Framework produces a self-stabilizing overlay network algorithm for* Skip+ *graphs that converges in* $O(\log n)$ *rounds, which is optimal.*

First note that the Transitive Closure Framework can produce a legal Skip+ graph in $O(\log n)$ rounds. This is faster than the original self-stabilizing Skip+ graph algorithm, which converged in $O(\log^2 n)$ rounds. Obviously, the Transitive Closure Framework manages to lower run-time complexity by trading space – specifically, it causes a $\Theta(n)$ increase in node degree. However, this is equivalent to the worst-case performance of some nodes when using the self-stabilizing Skip+ graph construction in [4].

5. The Local Repair Framework

The Transitive Closure Framework acts like a sledgehammer, initiating a full network rebuild even if the configuration is only minorly perturbed. However, there may be faulty configurations that are repairable efficiently, requiring action by only a small number of nodes (compared to n with TCF), and requiring only a limited amount of space (compared to $\Omega(n)$ for TCF). We call such faulty configurations "locally repairable".

Our local repair procedure has two components: identifying configurations that are locally repairable, and executing the actions to repair these configurations. We identify two components for our framework: a subroutine `L-REPAIR` and a local predicate defining the (potentially) locally-repairable network configurations `L-DETECT`. Notice that detectors now include nodes where `L-DETECT` is true.

Definition 3. *Let* $\texttt{L-DETECT}_u$ *be a predicate evaluated over* (λ^2, E_u) *at a node* u. $\texttt{L-DETECT}_u$ *is true when node* u *believes its neighborhood is part of a locally-repairable configuration – i.e. executing the* `L-REPAIR` *subroutine will result in a correct configuration.*

Definition 4. *Let* `L-REPAIR` *be a subroutine with parameters node* u*, a set* $\lambda = (V, \texttt{id}, \texttt{rs})$*, and a set of edges* E*.* `L-REPAIR` *returns a neighborhood for node* u*.* `L-REPAIR` *satisfies the following two properties:*

- *Convergence : If* $N(u)$ *is set to* $\texttt{L-REPAIR}(u, \lambda^2, E_u)$ *whenever* $\texttt{L-DETECT}_u$ *is true, after a finite number of rounds* $\texttt{L-DETECT}_u$ *becomes false.*
- *Connectivity : Setting* $N(u) = \texttt{L-REPAIR}(u, \lambda^2, E_u)$ *does not disconnect the network.*

Notice that, for some class of faults (which we shall call `LocalFault`), `L-REPAIR` is guaranteed to converge to a configuration where $\texttt{L-DETECT}_u$ is false.

This configuration may be correct or faulty – i.e. either the correct configuration is built or a detector u exists in the network with $\texttt{L-DETECT}_u = false$ (meaning u begins the Transitive Closure process). We call the maximum number of rounds required for this to happen the *recovery time* of `LocalFault` $RT_{\texttt{LocalFault}}(n)$.

Notice that when $\texttt{L-DETECT}_u = false$, our LRF reduces to the simple TCF, and $RT_{\texttt{LocalFault}}(n) = 0$ (as no local repairs are recognized). Similarly, when $\texttt{L-DETECT}_u = true$, all network configurations are stabilized using `L-REPAIR` in $RT_{\texttt{LocalFault}}(n)$ rounds (this is the approach taken for prior self-stabilizing overlay networks). In our work, we focus on cases where $\texttt{L-DETECT}_u$ is somewhere between $true$ and $false$.

5.1. The Local Repair Program

The Local Repair Framework (LRF) is shown in Program 2. Each node u evaluates $\texttt{L-DETECT}_u$ and, if $\texttt{L-DETECT}_u$ is true, executes `L-REPAIR`. If node u is a detector and $\texttt{L-DETECT}_u$ is false, then the Transitive Closure process is initiated as before.

Program 2 Local Repair Framework for Process u

Variables: neighborhood $N(u)$
in each round do
1. Send $N(u), \texttt{id}_u, \texttt{rs}_u$ to every neighbor $v \in N(u)$
2. Receive $N(v), \texttt{id}_v$, and $\texttt{rs}_v$ from each $v \in N(u)$
3. Compute from this information: $\lambda^2 := (N^2(u), \texttt{id}_u^2, \texttt{rs}_u^2)$ and
   $E_u := \{(u,v) | v \in N(u)\} \cup \{(v,w) | v \in N(u)\}$
4. **if** $\texttt{L-DETECT}_u(\lambda^2, E_u)$ **then**
5. $\quad N(u) := \texttt{L-REPAIR}(u, \lambda^2, E_u)$;
6. **else if** $N^2(u) \neq ON(N^2(u))$ **then**
7. $\quad$ Execute Lines 4-13 of Program 1;
8. **fi**

od

5.2. Proof of Local Repair

Proving that Program 2 provides a self-stabilizing overlay network framework is straightforward.

Lemma 2. *Starting from any configuration G_λ where $\forall u \in V$: $\texttt{L-DETECT}_u = false$, after at most* $\textsc{DetDiam}_{ON}(n) + \log n + 1$ *rounds the correct network $ON(\lambda)$ is constructed.*

Proof. Clearly if $G_\lambda = ON(\lambda)$, the lemma holds. If the network is faulty and $\texttt{L-DETECT}_u$ is false for all nodes, then at least one node will initiate the Transitive Closure process. The network remains connected, as neither the Transitive Closure process nor the `L-REPAIR` subroutine disconnects the network (by definition). Also notice that once the Transitive Closure process is initiated, it

continues until the correct network is built (as the network is locally checkable, at least one node will be a detector until the network is correct). The remainder of the proof follows from Theorem 1. □

Lemma 3. *Starting from any configuration where* $\texttt{L-REPAIR}_u = true$ *for all detectors, then in at most* $RT_{\texttt{LocalFault}}(n)$ *rounds either the correct overlay network is built or the Transitive Closure program is initiated.*

Proof. When $\texttt{L-REPAIR}_u = true$ for all detectors, the only neighborhood changes in the network occur due to line 5, which executes the `L-REPAIR` subroutine. By definition, after at most $RT_{\texttt{LocalFault}}(n)$ rounds the network is either correct, or at least one node has started the Transitive Closure program (line 7). □

Theorem 4. *Program 2 is a self-stabilizing overlay network construction algorithm that can recover from* any *configuration in at most* $RT_{\texttt{LocalFault}}(n) + \textsc{DetDiam}_{ON}(n) + \log n + 1$ *rounds. Furthermore, a subset of configurations are repairable in at most* $RT_{\texttt{LocalFault}}(n)$ *rounds.*

Proof. This follows easily from Lemmas 2 and 3. □

6. Example: `JOIN` in `SKIP+` Graphs

As overlay network membership is expected to be dynamic, accommodating nodes being added to the system is a commonly-addressed concern in overlay network algorithms. A node begins the Join process by connecting to a single node that is already a member of the current (correct) network. The goal of Join algorithms are to integrate the node into the correct network within some efficient amount of time. In this section, we instantiate our LRF by adding node Joins to Skip+ graphs.

6.1. The `L-DETECT` *Predicate and* `L-REPAIR` *Subroutine for* Skip+ Join

We begin by defining the following predicates that are evaluated locally on each node u, which together form $\texttt{L-DETECT}_u$. For ease of notation, let $pre(s_1, s_2)$ return the prefix match between strings s_1 and s_2, and $|pre(s_1, s_2)|$ be the length of the matching prefix. Recall that $E_u = \{(u, v) | v \in N(u)\} \cup \{(v, w) | v \in N(u)\}$, $N^2(u)$ is the set of the "at-most 2-hop neighbors" of u, and $\texttt{id}_u$ and $\texttt{rs}_u$ are the restrictions on `id` and `rs` from the set $N^2(u)$.

1. $SingleNodeFault_u(v) := [N_{\textsc{Skip+}(N^2(u), \texttt{id}_u, \texttt{rs}_u)}(u) \neq N(u)] \wedge$
   $[(E_u \setminus \{(v, x), \forall x\}) = \textsc{Skip+}(N^2(u) \setminus \{v\}, \texttt{id}_u, \texttt{rs}_u)]$
   - Node u detects its neighborhood is incorrect. However, node u's local neighborhood is consistent with the correct overlay network node u would see with node v removed.
2. $AllConnected_u := |N(u)| > 1 \implies \forall v \in N(u) : \exists w \neq v \in N(u)$ s.t. $w \in N(v)$

- If the neighborhood size is greater than 1 (node u has more than 1 neighbor), then for all neighbors $v \in N(u)$, there exists another neighbor $w \in N(u)$ (not equal to v) such that an edge exists between v and w.

3. $LongerMatchExists_u := \exists w \in (N^2(u) \setminus N(u))$ s.t. $\forall v \in N(u) : |pre(u.rs, w.rs)| > |pre(u.rs, v.rs)|$
   - In node u's 2-neighborhood, there exists a node w who has a longer matching random sequence than any node currently in u's immediate neighborhood.
4. $InitiatingJoin_u := LongerMatchExists_u \wedge AllConnected_u \wedge SingleNodeFault_u(u) \wedge [\forall v \in N(u) : \nexists w \in N(u)$ s.t. $|pre(u.rs, v.rs)| = |pre(u.rs, w.rs)|]$
   - For u, $LongerMatchExists_u$, $AllConnected_u$, and $SingleNodeFault_u(u)$ (see above) and no two nodes in $N(u)$ have the same length prefix match with u.
5. $CreatingJoinLinks_u := AllConnected_u \wedge \neg LongerMatchExists_u \wedge SingleNodeFault_u(u) \wedge (\exists w \in N^2(u)$ s.t. $(u, w) \in N_{\text{SKIP+}(N^2(u), \texttt{id}_u, \texttt{rs}_u)}(u) \wedge w \notin N(u))$
   - $AllConnected_u$, $\neg LongerMatchExists_u$, and $SingleNodeFault_u(u)$ are true. Also, node u has nodes in its 2-neighborhood that should be neighbors in the correct SKIP+ graph, yet are not neighbors now.
6. $JoinCompleted_u := AllConnected_u \wedge SingleNodeFault_u(u) \wedge N_{\text{SKIP+}(N^2(u), \texttt{id}_u, \texttt{rs}_u)}(u) \subset N(u)$
   - $AllConnected_u$ and $SingleNodeFault_u(u)$ is true for node u, and the neighbors of u are a superset of the nodes u should have in it's calculated ideal configuration.
7. $WaitForJoin_u := \exists v \in N(u) : SingleNodeFault_u(v) \wedge N_{\text{SKIP+}(N^2(u), \texttt{id}_u, \texttt{rs}_u)}(v) \neq N(v)$
   - Node u has a neighbor whose removal would make the network u is aware of correct, and $N(v)$ does not match the calculated neighorhood for v.
8. $JoinDetected_u := \exists v \in N(u) : SingleNodeFault_u(v) \wedge N_{\text{SKIP+}(N^2(u), \texttt{id}_u, \texttt{rs}_u)}(v) = N(v) \wedge N_{\text{SKIP+}(N^2(u), \texttt{id}_u, \texttt{rs}_u)}(u) \subset N(u)$
   - Node u has a neighbor v whose removal would make node u's neighborhood correct, node u's current neighborhood is currently a superset of u's calculated neighborhood, and node v has a correct neighborhood.

The next step in our framework is to define $\texttt{L-DETECT}_u$.

Definition 5. *For* JOIN *in* SKIP+ *graphs, let* $\texttt{L-DETECT}_u := InitiatingJoin_u \vee CreatingJoinLinks_u \vee JoinCompleted_u \vee WaitForJoin_u \vee JoinDetected_u$

We define the subroutine `L-REPAIR` for JOIN in SKIP+ graphs below. The high-level idea of our joining algorithm is simple. A joining node u traverses the network (by adding edges), searching for the node with the longest prefix match with u's random sequence ($InitiatingJoin_u$). Once this node is found, node u adds the appropriate neighbors for each level ($CreatingJoinLinks_u$). Once all neighbors have been added, node u deletes the edges that it used to find the longest prefix match. Other nodes in the network do nothing if there is a single-node fault in the network ($WaitForJoin_u$) until that single node has successfully joined the network ($JoinDetected_u$), and then their neighborhoods are updated to incorporate the new node.

Program 3 The `L-REPAIR` Subroutine for JOIN in SKIP+ Graphs

Parameters: Node u, set λ^2, edges E_u

1. **if** $InitiatingJoin_u$ **then**
   `// add the longest random sequence match to the neighborhood`
2. `return` $N(u) \cup \{x : x \in (N^2(u) \setminus N(u)) \wedge$
   $|pre(u.rs, x.rs)| = \max_{t \in N^2(u) \setminus u}(|pre(u.rs, t.rs)|) \wedge$
   $(\forall y \in (N^2(u) \cap \{z : |pre(u.rs, z.rs)| = |pre(u.rs, x.rs)|\}) :$
   $x.id \geq y.id)\}$;
3. **else if** $CreatingJoinLinks_u$ **then**
   `// add links level-by-level`
4. `return` $N(u) \cup \{x : x \in N_{\text{SKIP+}(N^2(u), \texttt{id}_u, \texttt{rs}_u)}(u) \wedge$
   $|pre(u.rs, x.rs)| = \max_{t \in N^2(u) \setminus N(u)}(|pre(u.rs, t.rs)|)$;
5. **else if** $JoinCompleted_u$ **then**
   `// delete edges used to find longest matching random`
   `// sequence, making the network correct for u`
6. `return` $N_{\text{SKIP+}(N^2(u), \texttt{id}_u, \texttt{rs}_u)}(u)$;
7. **else if** $JoinDetected_u$ **then**
8. `return` $N_{\text{SKIP+}(N^2(u), \texttt{id}_u, \texttt{rs}_u)}(u)$; `// incorporate the joined node`
9. **else** // $WaitForJoin_u$
10. `return` $N(u)$ `// do nothing`
11. **fi**

6.1.1. An Example JOIN

To clarify the JOIN procedure, we present an example JOIN for the SKIP+ graph from Fig. 2. Imagine a node u with $u.id = 27$ and $u.rs = 110$ joins the network by connecting to node w, with $w.id = 1$. Both node w and u will detect their local neighborhoods would appear correct if node u was removed ($SingleNodeFault_w(u)$ and $SingleNodeFault_u(u)$). Node w evalutes $WaitForJoin_w$ to true and will not change its neighborhood (Line 10), while node u evaluates $InitiatingJoin_u$ to true, since it has in its 2-neighborhood a node with a longer matching random sequence than any current neighbor (node x, with $x.id = 4$ and $x.rs = 111$). As a result, node u will, in one round, add the

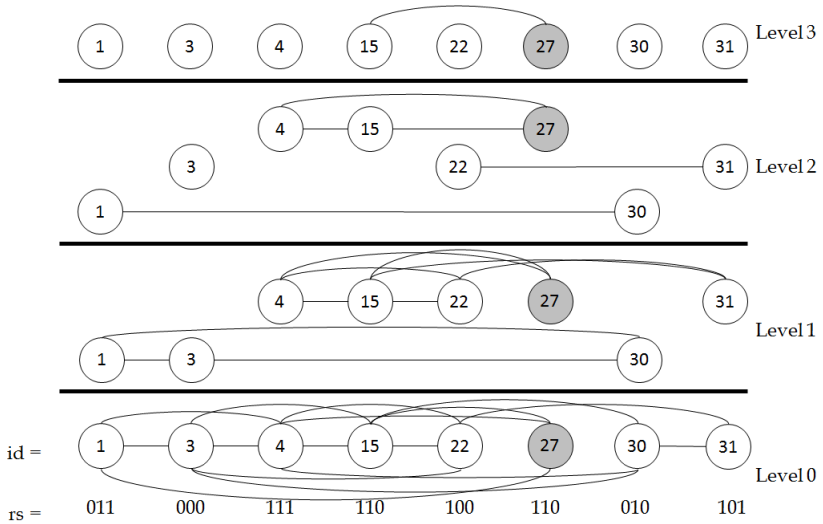

Figure 3: A Skip+ Join in progress. Node 27 is joining the network.

edge (u, x) (Line 2). In the next round, nodes u, w, and x will detect their local neighborhoods would appear correct if u was removed. Nodes w and x evaluate $WaitForJoin$ to true and do not change their neighborhoods (Line 10), while node u evaluates $InitiatingJoin_u$ to true and adds node y to its neighborhood (Line 2), with $y.id = 15$ and $y.rs = 110$. In the next round, nodes u, w, x, and y detect the network would appear correct if u was removed, and u can no longer find a node with a longer matching prefix than y. The system state at this point is shown in Figure 3.

Now that node u does not detect a node with a longer prefix match in its 2-neighborhood, it begins adding neighbors level-by-level ($CreatingJoinLinks_u$ evaluates to true). Node u already has the necessary links for levels 3 and 2 (created during the search for the longest match). All nodes that should be neighbors with u in level 1 are in u's 2-neighborhood. Therefore, node u will create a link in the next round to nodes a and b, with $a.id = 22$ and $b.id = 31$ (Line 4). Similarly, in the next round u will add links to c and d, with $c.id = 3$ and $d.id = 30$, filling in level 0. Neighbors of u during this process will evaluate $WaitForJoin$ to true (since u's neighborhood is not correct), and will not change their neighborhoods (Line 10).

Node u's neighborhood is now a superset of its calculated correct neighborhood (only the edge to node w, $w.id = 1$, is unnecessary). Node u, in the next round, will delete its edge to node w, since $JoinCompleted_u$ is true (Line 6). At this point the network is correct, and the Join procedure is completed.

6.2. Analysis of SKIP+ *Graph* JOIN

Lemma 4. *(Connectivity)* `L-REPAIR`*, when starting from a weakly-connected topology, does not disconnect the network.*

Proof. Notice that only lines 6 and 8 may delete edges from the network. Therefore, we only need consider these two actions when examining connectivity. First, consider a node u that deletes edges by executing line 6 of Program 3. The predicate $JoinCompleted_u$ must be true. Notice that for all $v \in N(u)$, $JoinDetected_v$ is false, as u's neighborhood contains too many neighbors, which all nodes v detect. Therefore, only node u will delete edges from its neighborhood when executing line 6. Notice that $AllConnected_u$ is true, meaning all of u's neighbors are connected to each other. Therefore, since only node u's neighborhood can change, and all $v \in N(u)$ are connected through paths not including u, the network remains connected when a node executes line 6.

Next, consider the case where node u deletes edges from its neighborhood using line 8 from Program 3. Notice that no node $v \in N(u)$ can delete an edge using line 6 (see prior paragraph). If an edge is deleted in line 8, then, it must be because the range of u has shrunk due to the presence of some $w \in N(u)$ (since $N(u)$ is correct if w is removed, and adding nodes can only shrink a calculated range). If $w \in N(u)$ causes some $range_i(u)$ to shrink, then all nodes that were neighbors in level i before are now at most distance 2 from u, and are reached by traveling through w. Node w will not delete these edges in that round. Therefore, the network remains connected during execution of `L-REPAIR`. □

Lemma 5. *(Convergence) The recovery time* $RT_{\text{SKIP+JOIN}}(n)$ *for Program 3 is* $2 \cdot c \cdot \log n + 2$.

Proof. Again recall that we assume the random sequence is of length $c \cdot \log n$. We consider two cases. First, imagine a single node u is joining a legal SKIP+ graph by creating a link to some node x in the network. At this point, $InitiatingJoin_u$ may be true for at most $c \cdot \log n$ rounds, as every round where $InitiatingJoin_u$ is true, u adds an edge to a node v (line 2) such that $|pre(u.rs, v.rs)|$ is greater than $|pre(u.rs, w.rs)|$ for all $w \in N(u)$, and $|\texttt{rs}| = c \cdot \log n$.When $InitiatingJoin_u$ is false, then, node u has a neighbor w such that w has the longest matching random sequence prefix with u in the network.

If $InitiatingJoin_u$ is false, $CreatingJoinLinks_u$ can be true for at most $c \cdot \log n$ rounds. To see this, imagine the longest random sequence prefix match node u has is with a neighbor w, with $|pre(u.rs, w.rs)| = i$. When $CreatingJoinLinks_u$ is true, node u executes line 4, which continues to add calculated neighbors to $N(u)$. After at most $c \cdot \log n - i + 1$ such additions, node u has a neighbor w' such that $w' \in range_i(u)$ and $u \in range_i(w')$, as the diameter of the subgraph induced by all nodes x with $|pre(u.rs, x.rs)| = i$ is $c \cdot \log n - i + 1$ (every hop is at least one random sequence bit closer to the destination).

For every subsequent round k where $CreatingJoinLinks_u$ remains true, u adds links to its correct neighbors in level $i - k$. Note the correct neighbors for node u at level $t - 1$ are in the correct 2-neighborhood of u at level t. To

see this, consider the nodes inside u's (correct) range in level $(t-1)$, which we shall denote $N_{t-1}(u)$. Note that all nodes in $N_{t-1}(u)$ that have a random sequence prefix match of length at least t are immediate neighbors of u in level t. Similarly, nodes in $N_{t-1}(u)$ that do not match at least t bits of the random sequence prefix are at most distance 2, as they must be neighbors with at least one node whose random sequence differs in bit t, which then matches the first t bits of u's random sequence. This node must be at most distance 1 from u in level t. Therefore, in $c \cdot \log n - i + 1 + i - 1 = c \cdot \log n$ rounds, $CreatingJoinLinks_u$ is false, as node u will be neighbors with all nodes in $N_{\text{SKIP+}(N^2(u),\texttt{id}_u,\texttt{rs}_u)}(u)$.

Finally, in one round node u will trim the unnecessary edges that were used to find the longest matching random sequence prefix (line 6). In one additional round, all nodes with which u is joining with will trim their neighborhoods to be equal to their calculated ideal neighborhoods (line 8). At this point, the network is now in a correct SKIP+ graph configuration. Therefore, $RT_{\texttt{LocalFault}}(n) = (c \cdot \log n) + (c \cdot \log n) + 2 = 2 \cdot c \cdot \log n + 2$ rounds.

Next, consider the case where $\texttt{L-DETECT}_u$ is true for some u, but the network is not the result of a single node attempting to join. By similar arguments to the correct JOIN case, a node can only execute $2 \cdot c \cdot \log n + 2$ steps before it has built what it believes is the correct network. If at any point during this execution L-DETECT is false for a node, but the node detects a faulty configuration, the Transitive Closure program is initiated. □

Theorem 5. *The Local Repair Framework given in Program 2, instantiated with the* L-REPAIR *subroutine from Program 3 and predicate* $\texttt{L-REPAIR}_u$ *from Definition 5, is a self-stabilizing algorithm for* SKIP+ *graphs with convergence time for* any *configuration at most* $3 \cdot c \cdot \log n + \log n + 2$ *rounds.* JOIN*s occur in at most* $2 \cdot c \cdot \log n + 2$ *rounds.*

Proof. By Lemma 5, JOINs require $2 \cdot c \cdot \log n + 2$ rounds. Since it will require at most $2 \cdot c \cdot \log n + 1$ rounds to initiate the Transitive Closure process, which will then require $\log n + c \cdot \log n + 1$ rounds to complete, any initial configuration requires at most $3 \cdot c \cdot \log n + \log n + 2$ rounds. □

7. Conclusions

The Transitive Closure Framework provides a simple starting point for reasoning about overlay networks. Exploring this simple framework provided us insights which allowed the definition of the detector diameter and the fault-detector distance, both of which have significant importance when analyzing other self-stabilizing overlay network algorithms. We used the fault-detector distance to prove a lower bound on overlay network construction. This bound is the only one we are aware of in current research that generalizes to many overlay networks, and is easy to use for evaluating overlay network families.

The implications of these observations are quite important in determining future research directions. By providing a lower bound of stabilization time for a class of graphs (the first such bound in overlay network research), future work

can bypass work with "bad" topologies (those that simply cannot be stabilized quickly), and instead focus on networks where scalable convergence may be a possibility.

Our work focused on single-configuration overlay networks. Note that creating overlay networks that have several legal configurations can lead to a version of consensus in requiring all nodes to agree on which configuration is being built. Furthermore, many structured overlay networks *are* single-configuration, including Chord [8] and Skip graphs [7]. Therefore, we do not feel considering only single-configuration networks to be very limiting.

We propose to extend this technique to other topologies. Such examination may not only allow classification of desirable topologies, but also may provide insight into simplifying the proof of detector diameters and bounding the fault-detector distance from above. This work could further help researchers and practitioners determine starting points for overlay network construction. It may also help ease the analysis of network performance, as currently there is no good "rule of thumb" for calculating the detector diameter. A consistent approach for determining detector diameter would ease the proof process, which currently relies on intuition and insight into the studied topology. Examining other topologies may also present other measures which can be used to evaluate topologies on more than just worst-case convergence time (for instance, examining the work requirements defined in [9]).

Currently, self-stabilizing overlay networks rely on topologies that are locally checkable. To our knowledge, however, there are no rules for determining which topologies are locally checkable, and which are not. Even if a topology is not locally-checkable, transforming it into a locally-checkable topology is a non-trivial open problem. Also, the use of global checkability (such as in [10]) is hardly a viable approach - however, global checking may be the only alternative when the topology is not locally-checkable. These are both avenues of research made easier by our simple Transitive Closure Framework.

While many P2P systems today have the capability for a linear amount of memory usage, an open problem is to refine the Transitive Closure Framework to operate in not only a scalable convergence time, but also with a polylogarithmic amount of memory during stabilization. For instance, is it possible to build the Skip+ graph in both polylogarithmic time and space for even the worst-case scenario? To date, no algorithm capable of polylogarithmic time *and* space has been defined. Such a result would be a significant breakthrough concerning self-stabilizing overlay network implementation.

In this paper, we have presented a simple, easy-to-use framework for constructing a self-stabilizing overlay network. While its performance is on par with the current state of the art, it's strength lies in its simplicity. This simplicity allowed us to define worst-case performance bounds for a class of overlay networks, and analyze a complex topology in Skip+ graphs. Future examination of the Transitive Closure Framework could yield more efficient algorithms, as well as aid in the further understanding of self-stabilizing networks and their bounds.

References

[1] E. W. Dijkstra, Self-stabilizing systems in spite of distributed control, Commun. ACM 17 (1974) 643–644.

[2] J. Aspnes, Y. Wu, O($\log n$)-time overlay network construction from graphs with out-degree 1, in: E. Tovar, P. Tsigas, H. Fouchal (Eds.), Principles of Distributed Systems, volume 4878 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2007, pp. 286–300. 10.1007/978-3-540-77096-1_21.

[3] M. Onus, A. W. Richa, C. Scheideler, Linearization: Locally self-stabilizing sorting in graphs, in: ALENEX, SIAM, 2007, pp. 99–108.

[4] R. Jacob, A. Richa, C. Scheideler, S. Schmid, H. Täubig, A distributed polylogarithmic time algorithm for self-stabilizing skip graphs, in: PODC '09: Proceedings of the 28th ACM symposium on Principles of distributed computing, ACM, New York, NY, USA, 2009, pp. 131–140.

[5] S. Kniesburges, A. Koutsopoulos, C. Scheideler, Re-chord: a self-stabilizing chord overlay network, in: Proceedings of the 23rd ACM symposium on Parallelism in algorithms and architectures, SPAA '11, ACM, New York, NY, USA, 2011, pp. 235–244.

[6] D. Peleg, Distributed computing: a locality-sensitive approach, Society for Industrial and Applied Mathematics, 2000.

[7] J. Aspnes, G. Shah, Skip graphs, in: SODA '03: Proceedings of the fourteenth annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003, pp. 384–393.

[8] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for internet applications, SIGCOMM Comput. Commun. Rev. 31 (2001) 149–160.

[9] D. Gall, R. Jacob, A. Richa, C. Scheideler, S. Schmid, H. Täubig, Modeling Scalability in Distributed Self-Stabilization: The Case of Graph Linearization, Technical Report TUM-I0835, TU Munich, 2008.

[10] S. Katz, K. J. Perry, Self-stabilizing extensions for message-passing systems, Distrib. Comput. 7 (1993) 17–26.