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

Andrew Berns, Sukumar Ghosh[**], and Sriram V. Pemmaraju[***]

Department of Computer Science
The University of Iowa
14 MacLean Hall
Iowa City, Iowa, USA 52242
[andrew-berns,sukumar-ghosh,sriram-pemmaraju]@uiowa.edu

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 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 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) which presents optimal convergence time from any configuration, and requires only a $O(1)$ factor of extra memory for handling node Joins.

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 create 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

[*] An early version of this work appeared as a Brief Announcement in PODC 2010
[**] This work is supported in part by National Science Foundation grant CNS-0956780
[***] This work is supported in part by National Science Foundation grant CCF 0915543

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. As a "toy" example, 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 IDs or those in which a node has three or more neighbors. 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. For these reasons, *self-stabilization*[3] is extremely relevant in overlay network construction. Recently the design of self-stabilizing overlay networks has received considerable attention – examples include algorithms for constructing *double-headed radix trees* [2], the LINEAR network [6], 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 appropriate 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 $\mathcal{CONGEST}$ [7]. 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 $\text{Diam}(G)$. In the current model of computation, two efficiency metrics seem to make most sense [6]. The first is the traditional worst-case number of rounds needed to terminate or stabilize (i.e., reach the target overlay network), and the second is the the maximum increase in the degree 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.

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 $\text{Diam}(G)$, is small relative to $|V|$, whereas $\text{Diam}(G_f)$ may be much larger than $\text{Diam}(G)$. 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. 1.2. Assume for now that V' is non-empty). The *detector diameter* $D(n)$ is the maximum distance in G_f between a non-detector node (i.e., a node in $V \setminus V'$) and its closest detector and serves as a measure of how well "detectors" are distributed in G_f. We show that any algorithm obtained from TCF can transform G_f to G in $O(D(n) + \log n)$ rounds, where $n = |V|$. In other words, the *stabilization time* of algorithms obtained from the TCF is $O(D(n) + \log n)$ rounds. To place this in context, we then show a lower bound: the worst-case self-stabilization time of an algorithm derived from TCF is bounded below by $\Omega(\mathrm{Diam}(G))$. The natural question to ask is: what is the gap between the lower bound $\Omega(\mathrm{Diam}(G))$ and the upper bound $O(D(n)+\log n)$? It may seem as though $D(n)$ can be as large as $\mathrm{Diam}(G_f))$, which, as we mentioned earlier, can be much larger than $\mathrm{Diam}(G)$. For a wide variety of overlay networks, we show that the detector diameter $D(n)$ is no more than $\mathrm{Diam}(G) + 1$, thus showing that the self-stabilization time of algorithms derived from TCF is within an additive logarithmic-factor of the optimal time.

– 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 $O(D(n) + \log n)$ rounds, 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.

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 [1]. 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 Model

Let V be a set of nodes. We suppose that there are two functions $\mathtt{id} : V \to \mathbb{Z}^+$ and $\mathtt{rs} : V \to \{0, 1\}^*$ that associate with each node in V a unique identifier and a random bit string. 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 $\lambda = (V, \mathtt{id}, \mathtt{rs})$ and $\mathcal{G}$ is the set of all directed graphs. In other words, the family of overlay networks associates a unique directed graph $ON(\lambda) \in \mathcal{G}$ with each labeled set $\lambda = (V, \texttt{id}, \texttt{rs})$ of nodes. At this point, the only assumption we make about ON is that it is well-defined for every member $\lambda \in \Lambda$.

Let E be an arbitrary set of directed edges on V such that the graph $G = (V, E)$ is weakly connected. Our goal is to design an algorithm that starts on any given labeled directed 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 out-neighbors $N(v)$, as part of its local state. $N(v)$ is node v's view of the network that it is part of. Initially, the sets $N(v)$, for all nodes $v \in V$ induce E (the input set of directed 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, updates $N(v)$ if necessary, and send out messages to all the nodes in $N(v)$. Message sizes are assumed to be unbounded and typically in each round the messages sent by v consist of the `id`-values and `rs`-values of all the nodes $N(v)$. Note that throughout the algorithm, node v can communicate with all of its current out-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 $\mathcal{CONGEST}$ [7].

Let $N_\lambda(v)$ denote the set of out-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 out-neighbors $N(v)$ is not equal to $N_\lambda(v)$. 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 may not know that it is faulty. This is because v is only aware of its current 2-neighborhood and within its 2-neighborhood everything may seem fine. More precisely, let V' denote the current 2-neighborhood of v, let $\texttt{id}'$ and $\texttt{rs}'$ be the restrictions of `id` and `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 identical to 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.

1.3 SKIP+ Graphs

In this paper, we use SKIP+ graphs to illustrate the utility of TCF. We define SKIP+ graphs in this section. SKIP graphs were first introduced in 2003 by Aspnes and Shah [1] 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, $u.rs$. 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 i bits of $u.rs$
- $pred(u, W)$: for any node u and subset W of nodes, $pred(u, W)$ is the node in the set W with largest id whose id is less than $u.id$.
- $succ(u, W)$: 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 $u.id$.

A legal SKIP graph consists of levels labeled $0, 1, 2, \ldots$. At each level i, a node u is neighbors with at most two nodes: $pred(u, \{w | pre_i(w) = pre_i(u)\})$ and $succ(u, \{w | pre_i(w) = pre_i(u)\})$, assuming such nodes exist. It is shown in [1] 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. 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 the leftmost node (`id` 1) and the rightmost node (`id` 22) nodes in Level 1. However, these two nodes are unware of the existence of each other and other nodes in the vicinity are unable to help.

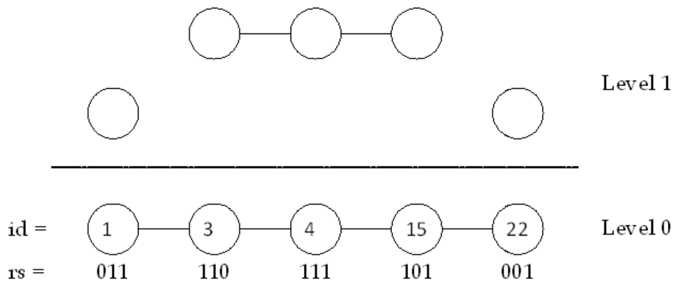

Fig. 1. An illegal SKIP graph in which none of the nodes are able to detect a fault.

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+ 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 a few 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.

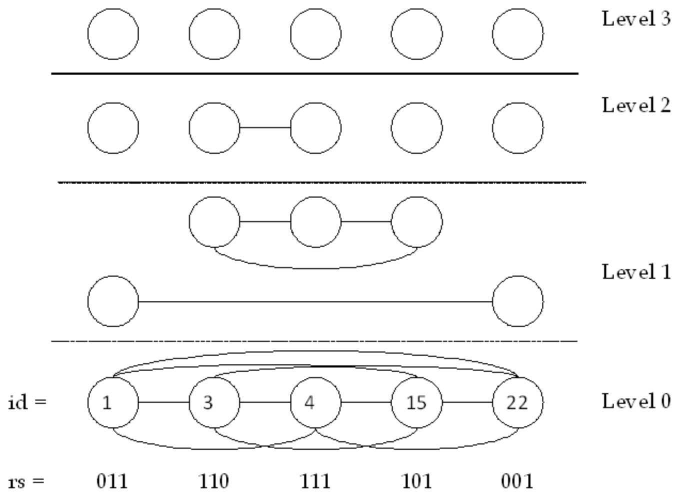

Fig. 2. A Skip+ Graph

- 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,
  $high_i(u) = \max\{succ_i(u, 0).id, succ_i(u, 1).id\}$.
- $range_i(u) = [low_i(u), high_i(u)]$.

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.

2 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 directed graph on the set of nodes V. Recall that $N(v)$ is the set of out-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 $\mathtt{id}$ and $\mathtt{rs}$ to $N(v)$. Let $N^2(v)$ denote the set of "at most 2-hop out-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 $\mathtt{id}$ and $\mathtt{rs}$ restricted to the set $N^2(v)$. Then $G_v^2 := ON(N^2(v), \mathtt{id}_v, \mathtt{rs}_v)$ 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. 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 λ^2, E_v, is true exactly when $E_v \subseteq E(G_v^2)$; 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 graph again [6]. Here, for a given $\lambda = (V, id, rs)$, G_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 $\lambda = (V, \texttt{id}, \texttt{rs})$, the* `REPAIR` *subroutine at node u sets the out-neighborhood of u to $N_\lambda(u)$ 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 10). $detect_u$ 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 9). In other words, each node with $detect = true$ in its neighborhood, expands its out-neighborhood to include all the out-neighbors of its out-neighbors (Line 9). Soon enough, every node has $detect = true$ and the network becomes a clique. At this point (which is detected in Line 5), the ideal neighborhood of each node can be built using the `REPAIR()` subroutine (Line 6).

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* $D(n)$ of any family ON of overlay networks as follows.

Fix a family of overlay networks ON. Let $\lambda = (V, \texttt{id}, \texttt{rs})$ be a labeled set of nodes and let E be an arbitrary set of directed edges on V such that $G = (V, E)$ is weakly 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 $D_\lambda(G)$, is the maximum hop distance in G between any node in V and the closest detector. The

Program 1 Transitive Closure Framework

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

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: $\lambda^2 := (N^2(v), \texttt{id}_v^2, \texttt{rs}_v^2)$ and
                                  $E_v := \{(v,u) | u \in N(v)\} \cup \{(u,w) | u \in N(v)\}$
4. $detect_u \leftarrow \texttt{DETECT}(\lambda^2, E_v) \vee detect_u$
5. if $detect_u \wedge \forall v \in N(u) : (detect_v \wedge (\{N(v) \cup v\} = \{N(u) \cup u\}))$ then
6.     $N(u) \leftarrow \texttt{REPAIR}(N(u) \cup u)$
7.     $detect_u \leftarrow false$
8. else if $detect_u \vee (\bigvee_{v \in N(u)} detect_v)$ then
9.     $N(u) \leftarrow N(u) \cup \{\bigcup_{v \in N(u)} N(v)\}$ //transitive closure
10.    $detect_u \leftarrow true$
11. fi
od
```

implication of this definition is that if the initial state of the system is network G, then some node v in G is $D_\lambda(G)$ hops from the closest detector and thus the TCF algorithm initiated by detectors requires $D_\lambda(G)$ rounds to reach v. The *detector diameter* $D(n)$ of a family ON of overlay networks is the maximum of $D_\lambda(G)$ over all $\lambda = (V, id, rs)$ with $|V| = n$ and all weakly connected, directed 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 $D(n)$ 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 locally-checkable family of overlay networks in at most* $D(n) + \log(n) + 1$ *rounds.*

The proof of this theorem is omitted from the paper due to space constraints and will appear in the full version of the paper. The intuition behind the upper bound is that it takes $D(n)$ rounds (in the worst case) for all nodes to realize that the network is faulty. Subsequently, all nodes are participating in the transitive closure process. This process reduces the diameter of the network by a factor of 2 in each round and as a result the network becomes a clique in an additional $\log n$ rounds.

2.1 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.

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

Proof. Let $\lambda = (V, \mathtt{id}, \mathtt{rs})$ with $|V| = n$. Let $G = ON(\lambda)$ and suppose that $d = \text{Diam}(G)$. There exists a shortest path consisting of distinct nodes $p_0, p_1, \cdots, p_d$ in the network G. Let $V' = V \setminus \{p_0\}$ and $\mathtt{id}'$ and $\mathtt{rs}'$ be restrictions of $\mathtt{id}$ and $\mathtt{rs}$ respectively to V'. Let $\lambda' = (V', \mathtt{id}', \mathtt{rs}')$ and $G' = ON(\lambda')$. There are two cases concerning the distance between nodes p_1 and p_d in G'.

Case 1: $dist_{G'}(p_1, p_d) > \frac{d}{2}$. If the distance between nodes p_1 and p_d remains at least $\frac{d}{2}$ in G', then we can insert node p_0 as a neighbor to node p_d. The network is now faulty, as p_0 must be a neighbor of p_1 (and vice-versa) in the ideal configuration. Furthermore, only p_d and its immediate neighbors have knowledge of node p_0, and these nodes are at least $\frac{d}{2}$ away from node p_1, a node that needs to change its local state. Therefore, the self-stabilization time from such a state is at least $\frac{d}{2}$.

Case 2: $dist_{G'}(p_1, p_d) < \frac{d}{2}$. Let nodes p_1 and p_d be closer than $\frac{d}{2}$ in G'. A node within $\frac{d}{2}$ of p_d must have then changed its neighborhood from G to G', or else the set of nodes within $\frac{d}{2}$ of p_d will not have changed, and p_1 remains at least $\frac{d}{2}$ from p_d. Notice, too, that all nodes within $\frac{d}{2}$ of p_d in G are also at least $\frac{d}{2}$ from p_1 in G. Therefore, if node p_0 is removed from G and the network is *not* reconfigured, only node p_1 and its immediate neighbors are detectors, and a faulty node exists at least $\frac{d}{2}$ away from p_1. Thus, removing p_0 from G results in a network from which stabilization takes at least $d/2$ rounds.

2.2 Bounding the Detector Diameter

Theorem 1 yields an $O(D(n) + \log n)$ upper bound on the self-stabilization time of TCF, whereas Theorem 2 yields an $\Omega(\text{Diam}(ON(n)))$ lower bound on the running time of any silent self-stabilizing algorithm. How close are these bounds to each other? In the following we show that for a wide variety of overlay networks $D(n)$ and Diam$(ON(n))$ are asymptotically identical implying that the upper bound is only an $O(\log n)$ additive factor larger than the lower bound. Another consequence of this is that for overlay networks for which Diam$(ON(n)) = \Omega(\log n)$, the upper and lower bounds are identical. We start by identifying families of overlay networks for which $D(n) = O(\text{Diam}(ON(n)))$ by stating an "axiom" they need to satisfy.

Axiom 1. ***[Subgraph Monotonicity]*** *Let $\lambda = (V, \mathtt{id}, \mathtt{rs})$ and $G = (V, E)$ be an arbitrary directed graph on V. For $u \in V$ and nonnegative integer k, let $B_k(u)$ be the set of nodes in G that are at most k hops from u in G. Let $\lambda_k(u) = (B_k(u), \mathtt{id}_k(u), \mathtt{rs}_k(u))$, where $\mathtt{id}_k(u)$ and $\mathtt{rs}_k(u)$ are the respective restrictions of $\mathtt{id}$ and $\mathtt{rs}$ to $B_k(u)$. If none of the nodes in $B_k(u)$ are detectors* *in G, then $G[B_k(u)]$ (i.e., the subgraph of G induced by $B_k(u)$) is identical to $ON(\lambda_k(u))$.*

Notice that many overlay networks satisfy subgraph monotonicity, including Skip+ and Linear networks. In fact, subgraph monotonicity may be an inherent property of locally checkable networks, although more investigation is required to support this possibility.

Theorem 3. *Let $\lambda = (V, \mathtt{id}, \mathtt{rs})$ and $G = (V, E)$ be an arbitrary directed graph on V. Let ON be a family of overlay networks satisfying Subgraph Monotonicity. Then $D_\lambda(G) \leq Diam(ON(n)) + 1$.*

This result is surprising in that Diam(n) may grow quite large compared to the size of Diam($ON(\lambda)$). However, increasing the diameter of a faulty configuration simply adds more detectors, meaning worst-case convergence time does not grow with Diam(n).

We can use Theorem 3 to find the stabilization bounds on certain overlay networks. Consider, for instance, the Linear graph, which was discussed in Sect. 2. Prior work in linearization has attempted to achieve a polylogarithmic convergence time [6]. However, one can easily prove that the Linear graph satisfies subgraph monotonicity. Therefore, Linear's convergence time is linear, and a polylogarithmic convergence time is impossible.

3 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.

3.1 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).

3.2 Analysis of the Transitive Closure Framework

To evaluate the performance of the TCF with regards to Skip+ graphs, we provide the detector diameter $D(n)$. The proof verifying this result will appear in a future full version of the paper.

Lemma 1. *The detector diameter for the family of* SKIP+ *graphs is* $D(n) = L + 1$*, where* $L = |\boldsymbol{rs}|$.

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

Theorem 4. *The Transitive Closure Framework produces a self-stabilizing overlay network algorithm for* SKIP+ *graphs that converges in* $L + \log(n)$ *rounds (where* L *is the length of the random sequence).*

First note that the Transitive Closure Framework can produce a legal SKIP+ graph in $O(\log n)$ rounds (when $L \in O(\log n)$). 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].

We can also use our above results to state that in fact our construction is within $O(\log n)$ of optimal (when $L \in O(\log n)$, the TCF algorithm is optimal).

Corollary 1. *By Theorems 2 and 4, the Transitive Closure Framework for* SKIP+ *graphs runs in* $O(\log n)$ *time, which is optimal.*

4 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).

Our local repair procedure uses a two-part approach: locally identifying those configurations that are locally repairable, and then executing the actions to repair these configurations. For a given λ, we identify two components for our framework: a program $Repair_\lambda^{ON}$ and a set of (potentially) locally-repairable network configurations $Repairable_\lambda^{ON}$. $Repairable_\lambda$ is defined in terms of a local predicate, so that locally repairable configurations may be detected as such. When ON is understood from the context, we drop ON and simply write $Repair_\lambda$ and $Repairable_\lambda$.

Definition 3. *Let* $CanRepair_u$ *be a predicate evaluated locally at each node* u*. Let* $Repairable_\lambda = \{G_\lambda : G_\lambda$ *is weakly connected and* $\forall v \in V : CanRepair_v\}$.

Definition 4. *Let* $Repair_\lambda$ *be a program executed by all nodes in* V*, and let* $Repair_\lambda^k[S]$ *represent the network resulting from executing* $Repair_\lambda$ *program* k *times, starting from configuration* S*.* $Repair_\lambda$ *satisfies the following properties:*

– *Convergence :* $\forall S \in Repairable_\lambda : Repair_\lambda^k[S] \notin Repairable_\lambda$*, for some finite* k

– *Connectivity : If $S \in Repairable_\lambda$ is weakly connected, then $Repair_\lambda[S]$ is also weakly connected*

Notice that $Repair_\lambda$ is guaranteed to converge to a configuration *not* in $Repairable_\lambda$. This configuration may be correct or faulty. The maximum number of rounds required for $Repair_\lambda$ to transform any n node network configuration in $G \in Repairable_\lambda$ to $ON(\lambda)$ is called the *repair time* $RT(n)$ of $Repair_\lambda$, while the maximum number of rounds required to transform any n node network configuration G to a faulty configuration $G' \notin Repairable_\lambda$ is called the *suppresion time* $ST(n)$ of $Repair_\lambda$, as it "suppresses" execution of the Transitive Closure process.

Notice that when $Repairable_\lambda = \emptyset$ ($CanRepair_u = false$), our LRF reduces to the simple TCF, and $RT(n)$ and $ST(n)$ are 0 (as no local repairs are recognized). Similarly, when $Repairable_\lambda = \mathcal{G}_\lambda$ ($CanRepair_u = true$), all network configurations are stabilized using $Repair_\lambda$ (this is the approach taken for prior self-stabilizing overlay networks). In our work, we focus on cases where $Repairable_\lambda$ is non-empty and does not include all configurations.

4.1 The Local Repair Program

The Local Repair Framework (LRF) is shown in Program 2. Each node u evaluates $CanRepair_u$ to see if its state is in $Repairable_\lambda$, and if so, initiates the local repair program $Repair_\lambda$. If the set is faulty and *not* in the $Repairable_\lambda$ set, then the Transitive Closure process is initiated as before.

Program 2 Local Repair Framework for Process u

```
Variables: neighborhood $N(u)$, predicate $CanRepair_u$
in each round do
1.  if $CanRepair_u$ then
2.      $Repair_\lambda^{ON}$;
3.  else if $N^2(u) \neq ON(N^2(u))$ then
4.      Begin executing the $TCF$;
5.  fi
od
```

Proving the correctness of LRF follows easily from the definitions and program given above. Due to space limitations, proofs of the following lemmas and theorem will appear in a future full version.

Theorem 5. *Program 2 is a self-stabilizing overlay network construction algorithm that can recover from any configuration in at most $ST(n)+D(n)+\log(n)+1$ rounds. Furthermore, a subset of configurations in $Repairable_\lambda$ will reach a correct network in $RT(n)$ rounds.*

5 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. As with prior research, we assume that a node begins the Join process by connecting to a single node that is already a member of the 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.

We begin by defining the following predicates that are evaluated locally on each node u, which we shall use to define $Repairable_\lambda$, $CanRepair_u$, and $Repair_\lambda^{\text{Skip+}}$ for Skip+ graphs. For ease of notation, let $pre(s1, s2)$ return the prefix match between strings $s1$ and $s2$, and let $|pre(s1, s2)|$ be the length of the matching prefix.

1. $AllConnected_u := |N(u)| > 1 \implies \forall v \in N(u) : \exists w \neq v \in N(u) s.t. v \in N(w) \wedge w \in N(v)$
2. $LongerMatchExists_u := \exists w \in (N^2(u) \setminus N(u)) s.t. \forall v \in N(u) : |pre(u.rs, w.rs)| > |pre(u.rs, v.rs)|$
3. $InitiatingJoin_u := LongerMatchExists_u \wedge AllConnected_u \wedge [\forall v \in N(u) : u \notin N(v) \wedge ON(N(v)) = N(v) \wedge (\nexists w \in N(u) s.t. |pre(u.rs, v.rs)| = |pre(u.rs, w.rs)|)]$
4. $CreatingJoinLinks_u := AllConnected_u \wedge \neg LongerMatchExists_u \wedge [\forall v \in N(u) : u \notin N(v)]$
5. $JoinCompleted_u := [N(u) = ON(N^2(u))] \wedge [\forall v \in N(u) : u \in N(v) \wedge ON(N^2(u)) \in N(v)]$
6. $JoinDetected_u := [\exists v \in N(u) : (N^2(u) \setminus \{v\}) = ON(N^2(u) \setminus v) \wedge (v \in ON(N(u))) \wedge (\forall x \in \{N(u) \setminus ON(N^2(u))\} : x \in N(v) \wedge v \in N(x))]$

The next step in our framework is to define the $CanRepair_u$, which will define the set $Repairable_\lambda$.

Definition 5. *For* Skip+ *graphs, let* $CanRepair_u := [(N^2(x) = ON(N^2(x))) \vee InitiatingJoin_x \vee CreatingJoinLinks_x \vee JoinCompleted_x \vee JoinDetected_x]$.

We define the $Repair_\lambda$ program for Joins in the Skip+ graph below. The high-level idea of our joining algorithm is simple. A joining node u traverses the network searching for the node with the longest prefix match with u's random sequence ($InitiatingJoin_u$). Once this node is found, node u adds 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, and makes the remaining correct edges correct. Nodes in the network will then detect the presence of a joined node, and make the appropriate changes ($JoinDetected_u$).

Theorem 6. *The Local Repair Framework given in Program 2, instantiated with the* $Repair_\lambda^{\text{Skip+}}$ *program from Program 3 and predicate* $CanRepair_u$ *from Definition 5, is a self-stabilizing algorithm for* Skip+ *graphs with convergence time for* any *configuration at most* $3 \cdot L + \log(n) + 3$ *rounds.* Join*s occur in at most* $2 \cdot L + 2$ *rounds, and require only* L *extra neighbors for the joining node.*

Program 3 $Repair_{\lambda}^{\text{SKIP+}}$ program for node u

```
Variables: neighborhood $N(u)$
in each round do
1.  if $InitiatingJoin_u$ then
        // search for best random sequence match
2.      $N(u) := N(u) + (x : x \in \{N^2(u) \setminus N(u)\} \wedge$
              $(\forall y \neq u \in N(u) : |pre(u.rs, x.rs)| > |pre(u.rs, y.rs)|))$;
3.  else if $CreatingJoinLinks_u$ then
4.      if $ON(N^2(u)) \in N(u)$ then
            // delete edges used to find longest matching random sequence
5.          $N(u) := N(u) \setminus \{x \in N(u) : x \notin ON(N^2(u))\}$;
6.          Make all remaining edges in $N(u)$ bidirectional;
7.      else
            // add links level-by-level
8.          $N(u) := N(u) \cup \{x : x \in ON(N^2(u)) \wedge$
              $|pre(u.rs, x.rs)| = max(|pre(u.rs, t.rs|), \forall t \in N^2(u) \setminus N(u)\}$;
9.      fi
10. else if $JoinDetected_u$ then
11.     $N(u) := ON(N^2(u))$; // incorporate the joined node
12. fi
od
```

References

1. Aspnes, J., Shah, G.: Skip graphs. In: SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms. pp. 384–393. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2003)
2. Aspnes, J., Wu, Y.: O($\log n$)-time overlay network construction from graphs with out-degree 1. In: OPODIS. pp. 286–300 (2007)
3. Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Commun. ACM 17(11), 643–644 (1974)
4. Jacob, R., Richa, A., Scheideler, C., Schmid, S., Täubig, H.: A distributed polylogarithmic time algorithm for self-stabilizing skip graphs. In: PODC '09: Proceedings of the 28th ACM symposium on Principles of distributed computing. pp. 131–140. ACM, New York, NY, USA (2009)
5. Kniesburges, S., Scheideler, C., Koutsopoulos, A.: Re-chord: A self-stabilizing chord overlay network. In: SPAA '11: Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures. ACM, New York, NY, USA (2011)
6. Onus, M., Richa, A.W., Scheideler, C.: Linearization: Locally self-stabilizing sorting in graphs. In: ALENEX. SIAM (2007)
7. Peleg, D.: Distributed computing: a locality-sensitive approach. Society for Industrial and Applied Mathematics (2000)