2005

Analysis of Link Reversal Routing Algorithms

Costas Busch
Rensselaer Polytechnic Institute

Srikanta Tirthapura
Iowa State University, snt@iastate.edu

Analysis of Link Reversal Routing Algorithms

Abstract
Link reversal algorithms provide a simple mechanism for routing in communication networks whose topology is frequently changing, such as in mobile ad hoc networks. A link reversal algorithm routes by imposing a direction on each network link such that the resulting graph is a destination oriented DAG. Whenever a node loses routes to the destination, it reacts by reversing some (or all) of its incident links. Link reversal algorithms have been studied experimentally and have been used in practical routing algorithms, including TORA [V. D. Park and M. S. Corson, A highly adaptive distributed routing algorithm for mobile wireless networks,in *Proc. INFOCOM*, IEEE, Los Alamitos, CA, 1997, pp. 1405--1413].

This paper presents the first formal performance analysis of link reversal algorithms. We study these algorithms in terms of *work* (number of node reversals) and the *time* needed until the network stabilizes to a state in which all the routes are reestablished. We focus on the *full reversal* algorithm and the *partial reversal* algorithm, both due to Gafni and Bertsekas [*IEEE Trans. Comm.*}, 29 (1981), pp. 11--18]; the first algorithm is simpler, while the latter has been found to be more efficient for typical cases. Our results are as follows: The full reversal algorithm requires $O(n^2)$ work and time, where n is the number of nodes that have lost routes to the destination. This bound is tight in the worst case.The partial reversal algorithm requires $O(n$ $\cdot$ $a^* + n^2)$ work and time, where a^* is a nonnegative integral function of the initial state of the network. Further, for every nonnegative integer α, there exists a network and an initial state with a^*=α, and with n nodes that have lost their paths to the destination, such that the partial reversal algorithm requires $\Omega(n\cdot {a^*} + n^2)$ work and time.There is an inherent lower bound on the worst-case performance of link reversal algorithms. There exist networks such that for *every* deterministic link reversal algorithm, there are initial states that require $\Omega(n^2)$ work and time to stabilize. Therefore, surprisingly, the full reversal algorithm is asymptotically optimal in the worst case, while the partial reversal algorithm is not, since a^* can be arbitrarily larger than n.

Keywords
link reversal routing, wireless networks, ad hoc networks, fault tolerance, self stabilization

Disciplines
Computer Sciences | Electrical and Computer Engineering

Comments
This article is from *SIAM Journal on Computing* 35 (2005): 305, doi: 10.1137/S0097539704443598. Posted with permission.

SIAM J. COMPUT.
Vol. 35, No. 2, pp. 305–326

ANALYSIS OF LINK REVERSAL ROUTING ALGORITHMS*

COSTAS BUSCH† AND SRIKANTA TIRTHAPURA‡

Abstract. Link reversal algorithms provide a simple mechanism for routing in communication networks whose topology is frequently changing, such as in mobile ad hoc networks. A link reversal algorithm routes by imposing a direction on each network link such that the resulting graph is a destination oriented DAG. Whenever a node loses routes to the destination, it reacts by reversing some (or all) of its incident links. Link reversal algorithms have been studied experimentally and have been used in practical routing algorithms, including TORA [V. D. Park and M. S. Corson, A highly adaptive distributed routing algorithm for mobile wireless networks, in *Proc. INFOCOM*, IEEE, Los Alamitos, CA, 1997, pp. 1405–1413].

This paper presents the first formal performance analysis of link reversal algorithms. We study these algorithms in terms of *work* (number of node reversals) and the *time* needed until the network stabilizes to a state in which all the routes are reestablished. We focus on the *full reversal* algorithm and the *partial reversal* algorithm, both due to Gafni and Bertsekas [*IEEE Trans. Comm.*, 29 (1981), pp. 11–18]; the first algorithm is simpler, while the latter has been found to be more efficient for typical cases. Our results are as follows:

- The full reversal algorithm requires $O(n^2)$ work and time, where n is the number of nodes that have lost routes to the destination. This bound is tight in the worst case.
- The partial reversal algorithm requires $O(n \cdot a^* + n^2)$ work and time, where a^* is a nonnegative integral function of the initial state of the network. Further, for every nonnegative integer α, there exists a network and an initial state with $a^* = \alpha$, and with n nodes that have lost their paths to the destination, such that the partial reversal algorithm requires $\Omega(n \cdot a^* + n^2)$ work and time.
- There is an inherent lower bound on the worst-case performance of link reversal algorithms. There exist networks such that for *every* deterministic link reversal algorithm, there are initial states that require $\Omega(n^2)$ work and time to stabilize. Therefore, surprisingly, the full reversal algorithm is asymptotically optimal in the worst case, while the partial reversal algorithm is not, since a^* can be arbitrarily larger than n.

Key words. link reversal routing, wireless networks, ad hoc networks, fault tolerance, self stabilization

AMS subject classifications. 68W15, 68W40, 68Q17, 68Q25, 68Q85

DOI. 10.1137/S0097539704443598

1. Introduction.

A mobile ad hoc network is a temporary interconnection network of mobile wireless nodes without a fixed infrastructure. The attractive feature of such a network is the ease with which one can construct it: there is no physical setup needed at all. If mobile nodes come within the wireless range of each other, then they will be able to communicate. More significant, even if two mobile nodes aren't within the wireless range of each other, they might still be able to communicate through a multihop path. However, the lack of a fixed infrastructure makes routing between nodes a hard problem. Since nodes are moving, the underlying communication graph is changing, and the nodes have to adapt quickly to such changes and reestablish their routes.

*Received by the editors May 8, 2004; accepted for publication (in revised form) May 12, 2005; published electronically October 7, 2005. A preliminary version of this paper has appeared in [2].
http://www.siam.org/journals/sicomp/35-2/44359.html

†Department of Computer Science, Rensselaer Polytechnic Institute, 110 8th Street, Troy, NY 12180 (buschc@cs.rpi.edu).
‡Department of Electrical and Computer Engineering, Iowa State University, 2215 Coover Hall, Ames, IA 50011 (snt@iastate.edu).

1.1. Link reversal. *Link reversal* routing algorithms [12, Chapter 8] are adaptive, self-stabilizing, distributed algorithms used for routing in mobile ad hoc networks. The first link reversal algorithms are due to Gafni and Bertsekas [7]. Link reversal is the basis of the temporally ordered routing algorithm TORA [10], and has also been used in the design of leader election algorithms for mobile ad hoc networks [9]. Link reversal routing is best suited for networks in which the rate of topological changes is high enough to rule out algorithms based on shortest paths, but not so high as to make flooding the only alternative.

Consider the graph representing the network, where the vertices are the wireless nodes, and each node has a link with each other node within its transmission radius. We assume that this underlying graph is undirected; i.e., the communication links are all bidirectional. Link reversal algorithms route on this graph by assigning directions to different links, hence converting it to a directed graph. A directed graph G is said to be connected if the underlying undirected graph of G (formed upon erasing the directions on the edges of G) is connected.

DEFINITION 1.1. *A connected directed acyclic graph with a single destination node is said to be* destination oriented *iff every directed path in the graph leads to the destination.*

For a given destination node, the link reversal algorithms assign directions to the links of this graph such that the resulting directed graph is a destination oriented directed acyclic graph (see Figure 1). Routing on a destination oriented network is easy: when a node receives a packet, it forwards the packet on *any* outgoing link, and the packet will eventually reach the destination.[1]

The task of the link reversal algorithm is to create and maintain the routes to the destination. When two nodes move out of each other's range, the link between them is destroyed, and some nodes might lose their routes to the destination. The routing algorithm reacts by performing *link reversals* (i.e., reorienting some of the edges) so that the resulting directed graph is again destination oriented. In particular, when a node finds that it has become a *sink* (has lost all of its outgoing links), then the node reacts by reversing the directions of some or all of its incoming links. The link reversals due to one node may cause adjacent nodes to perform reversals, and in this way, the reversals propagate in the network until the routes to the destination are reestablished.

Gafni and Bertsekas [7] describe a general family of link reversal algorithms and present two particular algorithms: the *full reversal* algorithm and the *partial reversal* algorithm (referred to as the GB algorithms in the rest of this paper). In the full reversal algorithm, when a node becomes a sink, it reverses the directions of all of its incident links. In the partial reversal algorithm, the sink reverses the directions of only those incident links that have not been recently reversed by adjacent nodes (a detailed description appears in the following section). The full reversal algorithm is simpler to implement, but the partial reversal algorithm may need fewer link reversals in some cases. Gafni and Bertsekas show that when link failures occur, these algorithms *eventually* converge to a destination oriented graph. However, it was not known how many reversals the nodes performed, or how much time it would take till convergence.

1.2. Performance of link reversal. We present the first formal performance analysis of link reversal routing algorithms. We give tight upper and lower bounds

[1]If there are multiple destinations in the network, then there is a separate directed graph for each destination; here, we will assume for simplicity that there is only one destination.

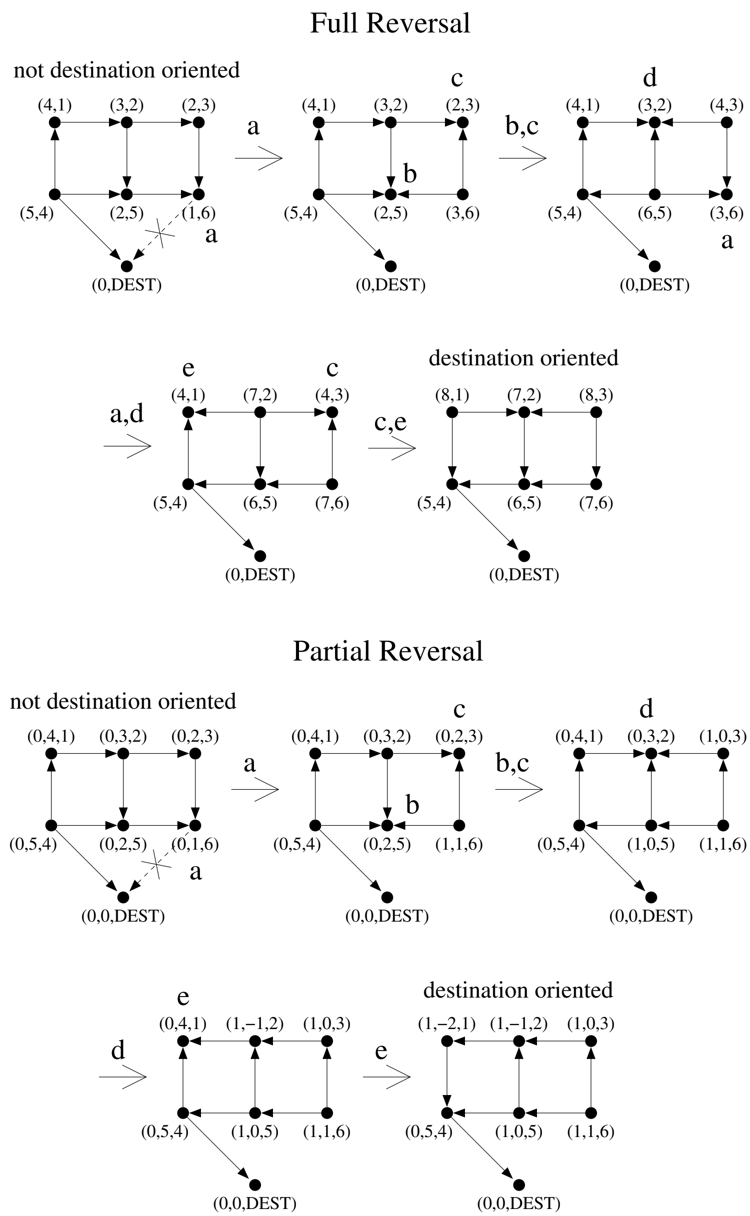

FIG. 1. *Sample executions of the GB full and partial reversal algorithms. Each transition is labeled with the nodes that reverse.*

on the performance of the full and partial reversal algorithms. We also show a lower bound on the performance of *any* deterministic link reversal algorithm. Surprisingly, from the perspective of worst-case performance, the full reversal algorithm is asymptotically optimal while the partial reversal algorithm is not.

Our setting for analysis is as follows. Suppose topological changes occur in the network, driving the system to a state in which some nodes have lost their paths to the destination. This is called the *initial state* of the network. If there are no further topological changes, the network is said to have *stabilized* when it again becomes destination oriented. We analyze two metrics:

Work: The number of *node reversals* until stabilization; a node reversal is the action of a sink reversing some or all of its adjacent links. This is a measure of the power and computational resources consumed by the algorithm in reacting to topological changes.

Time: The number of parallel steps until stabilization, which is a measure of the speed in reacting to topological changes. We model reversals so that each reversal requires one time step, and reversals may occur simultaneously whenever possible.

Reversals are implemented using *heights*. A reversal algorithm assigns a height to every node in the network. The link between adjacent nodes is directed from the node of greater height to the node of lesser height. A sink performs a reversal by increasing its height by a suitable amount. This will reverse the direction of some or all of its incident links. We consider *deterministic* link reversal algorithms, in which a sink increases its height according to some deterministic function of its own height and the heights of the adjacent nodes. The GB link reversal algorithms are deterministic.

In the analysis, we separate the nodes into bad and good. A node is *bad* if there is no route from the node to the destination. Any other node, including the destination, is *good*. Note that a bad node is not necessarily a sink. We present results for the following algorithms.

Full reversal algorithm. For the full reversal algorithm, we show that when started from an initial state with n bad nodes, the work and time needed to stabilize is $O(n^2)$. This bound is tight. We show that there are networks with initial states which require $\Omega(n^2)$ time for stabilization.

Our result for full reversal is actually stronger. For any network, we present a decomposition of the bad nodes in the initial state into *layers*, which allows us to predict *exactly* the work performed by each node in *any* distributed execution. A node in layer j will reverse exactly j times before stabilization. Our lower and upper bounds follow easily from the exact analysis.

Partial reversal algorithm. For the partial reversal algorithm, we show that when started from an initial state with n bad nodes, the work and time needed to stabilize is $O(n \cdot a^* + n^2)$, where a^* corresponds to the difference between the maximum and minimum heights of the nodes in the initial state. This bound is tight. We show that there are networks with initial states which require $\Omega(n \cdot a^* + n^2)$ time for stabilization.

The a^* value can grow unbounded as topological changes occur in the network. Consequently, in the worst case, the full reversal algorithm outperforms the partial reversal algorithm.

Deterministic algorithms. We show a lower bound on the worst-case work and time until stabilization for *any* deterministic reversal algorithm. We show that for any deterministic reversal algorithm on a given graph, there exists an initial state such that if a bad node d hops away from its closest good node, then it has to reverse d times before stabilization. Using this, we further show that there exist networks and initial states with n bad nodes such that the algorithm needs $\Omega(n^2)$ work and time until stabilization. As a consequence, from the worst-case perspective, the full reversal algorithm is work and time optimal, while the partial reversal algorithm is not.

Equivalence of executions. We show that for any deterministic reversal algorithm, all distributed executions of the algorithm starting from the same initial state are equivalent: (1) the resulting final state of the network upon stabilization is the same, and (2) each node performs the same number of reversals until stabilization in all executions. As a result, the work of the algorithm is independent of the execution schedule.

1.3. Related work. Link reversal algorithms were introduced by Gafni and Bertsekas in [7], where they provide a proof that shows that a general class of link reversal algorithms, including the partial and full reversal algorithms, eventually stabilize when started from any initial state. However, they do not give work and time bounds.

The TORA [10] builds on a variation of the GB partial reversal algorithm and adds a mechanism for detecting and dealing with partitions (disconnected components) in the network. The practical performance of the TORA has been studied in [11]. A variant of a link reversal routing algorithm is the lightweight mobile routing (LMR) algorithm [5, 6]. An overview of link reversal routing algorithms can be found in [12, Chapter 8]. A performance comparison of various ad hoc routing algorithms, including TORA, is presented in [1]. Further surveys can be found in [13, 14].

Malpani, Welch, and Vaida [9] build a mobility aware leader election algorithm on top of TORA and present partial correctness proofs (TORA does not have any) showing the stability of the algorithm. Although all the above work use link reversal, none of them have any formal performance analysis.

In the context of distributed sensor networks, coordination algorithms which are based on the paradigm of *directed diffusion* [8] are closely related to link reversal algorithms. For example, Intanagonwiwat et al. [8] state that their algorithm is closest to the TORA algorithm [10] in its attempt to localize the repairs due to node failures. Hence, our analysis also might lead to a better understanding of the performance of directed diffusion.

Link reversal algorithms attempt to always maintain routes to destinations in ad hoc networks. In cases when the network is sparsely populated with nodes, or when the rate of topology changes is too high, it may be infeasible to maintain such paths to the destination. In such cases, other strategies are needed for data delivery, such as those in [3, 4] which do not maintain paths to destination at all, but instead transmit data through strategies based on gossiping.

Outline of the paper. The rest of the paper is organized as follows. Section 2 contains a description of the GB partial and full reversal algorithms as well as a definition of deterministic algorithms. In section 3 we show the equivalence of executions of a given deterministic algorithm. Sections 4 and 5 contain the analyses of the full and partial reversal algorithms, respectively. In section 6, we show the general lower bound for deterministic link reversal algorithms, and we conclude with a discussion and open problems in section 7.

2. Link reversal algorithms. We assume that each node has a unique integer id and denote the node with id i by v_i. The nodes have heights which are guaranteed to be unique (ties broken by node ids) and are chosen from a totally ordered set. A link is always directed from the node of greater height to the node of the smaller height. The destination has the smallest height, and it is a special kind of sink which never reverses. Since any directed path in such a graph always proceeds in the direction of decreasing height, *the directed graph will always be a directed acyclic graph* (DAG). This is a significant feature, since the algorithms need not make further effort to maintain acyclicity in routing, and the graph remains acyclic even if topological changes occur.

If the underlying graph is connected, the link reversal algorithms bring the directed graph from its initial state to a state in which it is destination oriented. In our analysis, we consider only connected graphs. Note that there could possibly be multiple paths from any node to the destination. We now describe the two GB algorithms, adapting the discussion from [7], and then we define the class of deterministic algorithms.

Full reversal algorithm. In the full reversal algorithm, when a node becomes a sink it simply reverses the directions of all its incoming links (see top part of Figure 1, which is adapted from [7]). The algorithm can be implemented with heights as follows. The height h_i of node v_i is the pair (a_i, i) (the second field is used to break ties). The height of the destination (say v_d) is $(0, d)$. Heights are ordered lexicographically. If v_i is a sink, then its height upon reversal is updated to be larger than the heights of all its neighbors. Let $N(v_i)$ denote the set of adjacent nodes to v_i. Formally, the height of v_i after its reversal is $(\max\{a_j \mid v_j \in N(v_i)\} + 1, i)$.

Partial reversal algorithm. In the partial reversal algorithm, the height of each node v_i is a triple (a_i, b_i, i). As in full reversal, node v_i reverses only when it becomes a sink. The height of v_i after reversal is greater than the height of at least one neighbor, but may not be greater than the height of every neighbor. The height of the destination v_d is $(0, 0, d)$. Heights are ordered lexicographically. The second field b_i helps the sink avoid reversing links toward adjacent nodes, which have caused the node to become a sink in the first place. Thus, reversals are not immediately propagated to parts of the network which have already reversed. Formally, let $\bar{h}_i = (\bar{a}_i, \bar{b}_i, i)$ denote the height of v_i after its reversal. See the bottom part of Figure 1 (adapted from [7]) for an example execution of the partial reversal algorithm. The partial reversal algorithm updates heights as follows:

- $\bar{a}_i = \min\{a_j \mid v_j \in N(v_i)\} + 1$.
- $\bar{b}_i = \min\{b_j \mid v_j \in N(v_i) \text{ and } \bar{a}_i = a_j\} - 1$ if there exists a neighbor v_j with $\bar{a}_i = a_j$; otherwise, $\bar{b}_i = b_i$.

The basic idea behind these functions is as follows. In a network state I, where v_i is a sink, we can divide the neighbors of v_i into two categories: (i) $\{v_j | a_j = a_i\}$ and (ii) $\{v_j | a_j > a_i\}$. Node v_i must have reversed after the last reversal of every node in category (i) since, otherwise, those nodes would have $a_j \geq a_i + 1$. On the other hand, nodes of category (ii) must have reversed after the last reversal of node v_i since, otherwise, the heights of those nodes would not be higher than a_i. Therefore, node v_i is a sink in state I due only to nodes of category (ii). Thus, when v_i reverses after state I, its new height should be set so that the links point only toward nodes of category (i). This is achieved by setting $\bar{a}_i = a_i + 1$. In order to make the nodes of category (ii) to point to v_i, we need only take care of nodes with $a_j = \bar{a}_i$, for which we adjust $\bar{b}_i$ to be lower than all the corresponding b_j's.[2]

Deterministic algorithms. A deterministic reversal algorithm is defined by a "height increase" function g. We assume that the heights are chosen from some totally ordered universe and that the heights of different nodes are unique. If node v is a sink of degree k, whose current height is h_v, and adjacent nodes $v_1, v_2, \ldots, v_k$ have heights $h_1, h_2, \ldots, h_k$, respectively, then v's height after reversal is $g(h_1, h_2, \ldots, h_k, h_v)$. Function g is such that the sink reverses at least one of its incoming links. The GB full and partial reversal algorithms are deterministic.

[2]The function for the update of $\bar{a}_i$ is expressed in terms of the minimum value of the neighbors, since topological changes might generate a state in which no neighbor has $a_j = a_i$.

3. Equivalence of executions. In this section, we prove some properties about deterministic link reversal algorithms. The main result of this section is that for any deterministic reversal algorithm, all executions that start from the same initial state are essentially equivalent: the resulting final state of the network upon stabilization is the same. We actually show the stronger result that each node performs the same number of reversals, with the same heights, until stabilization in all executions. We first give some basic definitions for states and executions; then we define the dependency graph, which will help to show that all executions are equivalent, and finally, we give the main result.

3.1. States, executions, and dependency graphs. Here we give basic definitions for states and executions. At any configuration of the network, the *node state* of a node v is defined as the current height of v. The *network state* is defined as the collection of the individual states of all the nodes in the network. Note that the network state uniquely determines the directions of all the links in the network.

A node reversal r is defined as a tuple $r = (v, h, H)$, where v is the sink executing the reversal, h is v's height before reversal, and H is the set of the heights of all of v's neighbors before the reversal. Given an initial state containing bad nodes, an *execution* E is defined as a sequence of reversals $E = r_1, r_2, \ldots, r_k$, where $r_i = (v_i, h_i, H_i)$, and $1 \leq i \leq k$. A *complete execution* is defined as an execution that ends in a destination oriented graph; unless otherwise stated, we will refer to complete executions from here on.

Clearly, there are many possible executions starting from the same initial state. We give the following definition for equivalent executions.

Definition 3.1. *Starting from the same initial state, two executions are equivalent if they give the same final state.*

In order to show that two executions are equivalent, we will use the dependency graphs of the executions which we define next. Any execution imposes a partial order on the reversals. The partial order induced by execution $E = r_1, r_2, \ldots, r_k$ is defined as a directed graph whose nodes are the reversals r_i, $i = 1, \ldots, k$. There is a directed edge from $r_i = (v_i, h_i, H_i)$ to $r_j = (v_j, h_j, H_j)$ if

- v_j is a neighbor of v_i, and
- r_j is the first reversal of v_j after r_i in execution E.

We will refer to this graph as the *dependency graph* of execution E. Intuitively, if there is a directed path between reversals r_i and r_j in the dependency graph, then the order of these two reversals cannot be interchanged. Moreover, if there is no directed path from r_i to r_j, then these two reversals are independent and can be performed in parallel (in the same time step).

We define the *depth* of a reversal in the dependency graph as follows. A reversal that does not have any incoming edges has depth 1 (these are the reversals of the nodes which are sinks in the initial state). The depth of any other reversal r is one more than the maximum depth of a reversal which points to r. The depth of the dependency graph is the maximum depth d of any reversal in the graph. The dependency graph is important for the following reason.

Fact 1. The dependency graph of an execution uniquely determines

- the final state of the network,
- the number of reversals performed by each node, and
- the stabilization time when all sinks reverse simultaneously, which is the depth of the dependency graph d.

3.2. Proof of equivalence. We show that all executions of a link reversal algorithm give the same dependency graph, which implies that the executions are equivalent. Fact 1 implies that this result is actually stronger than simply showing that executions are equivalent. We first show two lemmas that will be of use in further proofs.

LEMMA 3.2. *For any reversal algorithm starting from any initial state, a good node never reverses until stabilization. Further, a good node always remains good until stabilization.*

Proof. If v is a good node, then by definition there exists a path $v = v_k, v_{k-1}, \dots, v_1$, $v_0 = s$, where s is the destination, and there is an edge directed from v_i to v_{i-1} for each $i = 1, \dots, k$.

For each $i = 0, \dots, k$, we prove that node v_i never reverses, using induction on i. The base case ($i = 0$) is obvious since the destination never reverses. Suppose the hypothesis is true for $i = l < k$. Then v_l never reverses, so that the edge between v_{l+1} and v_l is always directed from v_{l+1} to v_l. Thus, there is always an outgoing edge from v_{l+1}, which implies that v_{l+1} never reverses, and completes the proof by induction.

This also implies that the directed path $v = v_k, v_{k-1}, \dots, v_1, v_0 = s$ always exists in the network, showing that node v remains good. ∎

LEMMA 3.3. *If a node v is a sink, then v remains a sink until it reverses. Further, v eventually reverses.*

Proof. If a node v is a sink, then clearly none of its neighbors can be sinks at the same time, and hence they cannot reverse. Thus, the only node that can change the direction of the incoming links to v is v itself. Reversals by other nodes in the network do not affect this. Thus, v remains a sink until it reverses.

Further, the reversal of v is enabled continuously until v actually reverses. Since we assume that the distributed system eventually makes progress (an action that is continuously enabled will eventually take place), v eventually reverses. ∎

THEOREM 3.4 (identical dependency graphs). *All executions of a deterministic reversal algorithm starting from the same initial state give identical dependency graphs.*

Proof. Consider two executions of the algorithm starting from the same initial state, say, execution $R = r_1, r_2, \dots$ and execution $S = s_1, s_2, \dots$. Let p_R and p_S be the dependency graphs induced by R and S, respectively. We will show that p_R and p_S are identical.

We will show by induction that, for every $k = 1, 2, \dots$, the induced subgraph of p_R consisting of vertices at depths k or less is identical to the similarly induced subgraph of p_S consisting of vertices at depths of k or less.

Base case $k = 1$. Consider any reversal $r = (v, h, H)$ in p_R at depth 1. Since r does not have any incoming edges in p_R, node v must be a sink in the initial state of the network. From Lemma 3.3, v must also reverse in S. Since h and H are the heights of v and its neighbors, respectively, in the initial state, and they do not change until v reverses at least once, the first reversal of v in S is also (v, h, H), and is at depth 1. Similarly, any other reversal at depth 1 in p_S is also a reversal at depth 1 in p_R, and this proves the base case.

Inductive case. Suppose the hypothesis is true for all $k < l$. We show that it is also true for $k = l$. Consider any reversal $r = (v, h, H)$ at depth l in p_R. We show that this reversal is also present in p_S with the same set of incoming edges. Let U be the set of reversals that are pointing into r in p_R. Once all reversals in U are executed, node v becomes a sink in execution R. From the inductive step, all reversals in U are also present in p_S, and hence in S. We examine two cases.

Case 1. *Reversal r is the first reversal of v in R.* Then, the execution of all reversals in U will also cause v to be a sink in S. Thus v also will reverse in S. Its height before reversal in S is h, since the height has not changed from the initial state. Consider the heights of v's neighbors before v's reversal in S. These are equal to H. The reason is as follows. The neighbors of v who haven't reversed so far in S have the same height as in the initial state. The other neighbors are present in U, and hence their heights are the same as in H. Thus, there is a reversal (v, h, H) at depth l in p_S whose incoming edges are the same as in p_R.

Case 2. *Reversal r is not the first reversal of v in R.* Let r' denote the previous reversal of v in R. Since r' is at a lower depth in p_R than r, by the induction hypothesis, r' is also present in p_S. After reversal r', node v will be in the same state in both R and S. After the reversals in U, v's neighbors will be in the same state in S as in R. Thus, the reversal (v, h, H) is also present in S at depth l with the same incoming edges as in p_R.

Thus, we have shown that every node at depth l in p_R is present at depth l of p_S, with the same incoming edges. The same argument goes the other way too: every node in p_S is present in p_R. This proves the inductive case for $k = l$, and concludes the proof. ∎

The following corollary follows from Fact 1 and Theorem 3.4.

COROLLARY 3.5 (equivalence of executions). *All executions of a deterministic reversal algorithm starting from the same initial state are equivalent. Moreover,*

- *the number of reversals of each node in every execution is the same, and*
- *when all sinks reverse simultaneously, the stabilization time of every execution is d, the depth of the (unique) dependency graph.*

4. Full reversal algorithm.

In this section, we present the analysis of the full reversal algorithm. We present a decomposition of the bad nodes in the initial network state into *layers*, which allows us to predict *exactly* the work performed by each node in any distributed execution until stabilization: a node at layer i will reverse exactly i times. From the exact analysis, we obtain worst-case bounds for the work and time needed for stabilization.

4.1. State sequence for full reversal.

In order to obtain the exact analysis, we first show that, starting from any initial state, there exists an execution which consists of consecutive execution segments such that at each execution segment, each remaining bad node reverses exactly once. We will then use this result to determine the exact number of reversals of each bad node in the layer decomposition.

In particular, consider some initial state I_1 of the graph which contains bad nodes. We will show that there is an execution $E = E_1, E_2, E_3, \dots$, and states $I_1, I_2, I_3, \dots$, such that execution segment E_i, $i \geq 1$, brings the network from a state I_i to a state I_{i+1}, and in E_i each bad node of I_i reverses exactly one time. In order to show that E exists, we need to prove the following two lemmas.

LEMMA 4.1. *Consider a state I in which a node v is bad. Then, node v will reverse at least one time before it becomes a good node.*

Proof. If v is a sink, then clearly node v has to reverse at least one time. Now consider the case when v is not a sink in state I. Suppose, for contradiction, that node v becomes good without performing any reversals after state I. Consider an execution which brings the graph from state I to a state I^g in which node v is good. A *nonreversed* node is any node w such that in state I node w is bad, while in state I^g node w is good, and w does not reverse between I and I^g. Since in state I^g node

v is good, there must exist in I^g a directed path $v, v_1, \ldots, v_{k-1}, v_k$, $k \geq 1$, in which v_k is good in I^g and I.

We will show that nodes $v_1, \ldots, v_{k-1}$ are nonreversed. Consider node v_1. Assume for contradiction that node v_1 has reversed between states I and I^g. Since in I^g there is a link directed from node v to node v_1, and v_1 has reversed between states I and I_g, it must be that node v has reversed at least one time, a contradiction. Thus, node v_1 is nonreversed. Similarly, using induction, we can easily show that nodes $v_2, \ldots, v_{k-1}$ are also nonreversed. Since nodes $v_1, \ldots, v_{k-1}$ are nonreversed, it has to be that in state I there is a directed path $v, v_1, \ldots, v_{k-1}, v_k$. Thus, in state I node v is a good node. This contradiction completes the proof. ∎

LEMMA 4.2. *Consider some state I which contains bad nodes. There exists an execution which brings the network from state I to a state I' (not necessarily a final state) such that every bad node of state I reverses exactly one time.*

Proof. Suppose for contradiction that there is no such execution. Then, there exists an execution E^f which brings the system from state I to a state I^f such that the following conditions hold:

1. There is at least one bad node in I which hasn't reversed in E^f. Let A denote the set of such bad nodes of I.
2. Any other bad node v of I, with $v \notin A$, has reversed exactly one time. Let B denote the set of such bad nodes of I.
3. The number of nodes in set B is maximal.

First we show that all the nodes that are sinks in state I^f have to be members of set B. Suppose that a sink in state I^f is a member of set A. Then the sink hasn't reversed since state I. If the sink reverses then it could be an additional member of set B. Thus, B is not maximal as required by the third condition. Therefore, the sink has to be a member of B.

Next we show that at least one node in A is a sink in state I^f, which proves that execution E^f does not exist. Assume for contradiction that no node of A is a sink in I^f. Then, each node in A has an outgoing edge in I^f. These outgoing edges from A cannot point toward nodes in B, since the nodes in B have reversed their edges, while the nodes in A haven't. Moreover, these outgoing edges cannot point toward good nodes of state I, since this would imply that nodes in A are good in state I^f, while Lemma 4.1 implies that each node of set A remains bad in state I^f. Thus, these outgoing edges must point toward nodes in set A. Since each node in set A has an outgoing edge in set A, it must be, from the pigeonhole principle, that there is a walk in which a node in A is repeated. Thus, there is a cycle in the graph, violating the fact that the graph is acyclic. Thus, it must be that a node in A is a sink, a contradiction. ∎

Lemma 4.2 implies that the execution segments E_i and the states I_i exist. The *link-state* of a node v is the vector of directions of its incident links. We show that each execution segment leaves the link-state of bad nodes unchanged for the bad nodes, which are not adjacent to good nodes.

LEMMA 4.3. *If in state I_i, $i \geq 1$, node v is bad and v is not adjacent to a good node, then v will remain in the same link-state in I_{i+1}.*

Proof. Let $A(v)$ denote the set of nodes adjacent to v in state I_i. Since all nodes in $A(v)$ are bad in state I_i, each of them reverses in execution E_i. Moreover, v also reverses in E_i. These reversals leave the directions of the links between v and $A(v)$ in state I_{i+1} the same as in state I_i. ∎

4.2. Layers for full reversal. Here, we show that given some initial state I with bad nodes, it is possible to decompose the bad nodes into layers and determine the exact number of reversals for the nodes of each layer until stabilization: a node in layer i reverses exactly i times.

In particular, we decompose the bad nodes into layers $L_1^I, L_2^I, \ldots, L_m^I$, defined inductively as follows (see Figure 2). A bad node v is in layer L_1^I if the following conditions hold:

- There is an incoming link to node v from a good node, or
- there is an outgoing link from node v to a node in layer L_1^I.

A node v is in layer L_k^I, $k > 1$, if k is the smallest integer for which one of the following hold:

- There is an incoming link to node v from a node in layer L_{k-1}^I, or
- there is an outgoing link from node v to a node in layer L_k^I.

From the above definition, it easy to see that any node of layer L_k^I, where $k > 1$, can be connected only with nodes in layers L_{k-1}^I, L_k^I, and L_{k+1}^I. The nodes of layer L_1^I are the only ones that can be connected with good nodes. The links connecting two consecutive layers L_{k-1}^I and L_k^I can be directed only from L_{k-1}^I to L_k^I. Note that the number of layers m is not greater than the number of bad nodes in the network n.

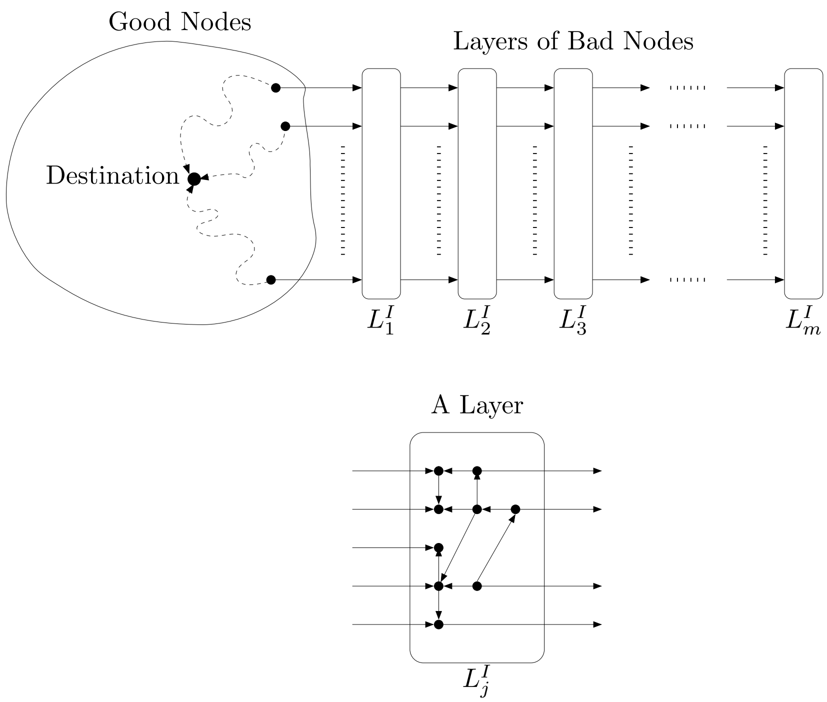

FIG. 2. *Decomposition of the bad nodes into layers.*

Consider now the states $I_1, I_2, \ldots$ and execution segments $E_1, E_2, \ldots,$ as described in section 4.1. For each of these states we can divide the bad nodes into layers, as described above. In the following sequence of lemmas we will show that the layers of state I_1 become good one by one at the end of each execution segment E_i, $i \geq 1$. We show now that the first layer of state I_i becomes good at the end of execution E_i.

LEMMA 4.4. *At the end of execution E_i, $i \geq 1$, all the bad nodes of layer $L_1^{I_i}$ become good, while all the bad nodes in layers $L_j^{I_i}$, $j > 1$, remain bad.*

Proof. First we show that the bad nodes of layer $L_1^{I_i}$ become good. There are two kinds of bad nodes in layer $L_1^{I_i}$ at state I_i: type α, nodes which are connected with an incoming link to a good node; and type β, nodes which are connected with an outgoing link to another node in layer $L_1^{I_i}$.

It is easy to see that there is a direct path from any β node to some α node, consisting of nodes of layer $L_1^{I_i}$. Since all bad nodes reverse exactly once in execution E_i, all α nodes become good in state I_{i+1}. Moreover, from Lemma 4.3, the paths from β nodes to α remain the same in state I_{i+1}. Thus, the β nodes also become good in state I_{i+1}. Therefore, all the bad nodes of layer $L_1^{I_i}$ become good in state I_{i+1}.

Now we show that the bad nodes in layers $L_j^{I_i}$, $j > 1$, remain bad in state I_{i+1}. From Lemma 4.3, in state I_{i+1}, the links connecting layers $L_1^{I_i}$ and $L_2^{I_i}$ are directed from $L_1^{I_i}$ to $L_2^{I_i}$. Thus, in state I_{i+1}, there is no path connecting nodes of layer $L_2^{I_i}$ to good nodes. Similarly, there is no path from the nodes of layer $L_j^{I_i}$, for any $j > 2$, to good nodes. Thus all nodes in layers $L_j^{I_i}$, $j > 1$, remain bad. $\square$

We now show that the basic structure of layers of the bad nodes remains the same from state I_i to state I_{i+1}, with the only difference being that the first layer of I_{i+1} is now the second layer of I_i.

LEMMA 4.5. $L_j^{I_{i+1}} = L_{j+1}^{I_i}$, $i, j \geq 1$.

Proof. From Lemma 4.4, at the end of execution E_i, all the bad nodes of layer $L_1^{I_i}$ become good, while all the bad nodes in layers $L_j^{I_i}$, $j > 1$, remain bad. From Lemma 4.3 all bad nodes in layers $L_j^{I_i}$, $j > 1$, remain in the same link-state in I_{i+1} as in I_i. Therefore, $L_j^{I_{i+1}} = L_{j+1}^{I_i}$, $j \geq 1$. $\square$

From Lemmas 4.4 and 4.5, we have that the number of layers is reduced by one from state I_i to state I_{i+1}. If we consider the layers of the initial state I_1, we have that all the bad nodes in the layers become good one by one at the end of executions $E_1, E_2, E_3, \ldots$ in the order $L_1^{I_1}, L_2^{I_1}, L_3^{I_1}, \ldots$. Since in each execution E_i all the bad nodes reverse exactly one time, we obtain the following.

LEMMA 4.6. *Each node in layer $L_j^{I_1}$, $j \geq 1$, reverses exactly j times before it becomes a good node.*

From Corollary 3.5, we know that all possible executions when started from the same initial state require the same number of reversals. Thus, the result of Lemma 4.6, which is specific to the particular execution E, applies to all possible executions. Therefore, we obtain the following theorem.

THEOREM 4.7 (exact number of reversals for full reversal). *For any initial state I, and any execution of the full reversal algorithm, $L_1^I, L_2^I, \ldots$ is a division of the bad nodes in I into layers such that each node in layer L_j^I, $j \geq 1$, reverses exactly j times before it becomes a good node.*

4.3. Worst-case bounds for full reversal. We now give worst-case upper and lower bounds for the work and time needed for stabilization by the full reversal algorithm. Both bounds are obtained with the use of Theorem 4.7.

From Theorem 4.7, we have that for any initial state I, each node in layer L_j^I reverses exactly j times until it becomes good. Thus, the total number of reversals of the nodes of layer j is $j \cdot |L_j^I|$. If there are m layers of bad nodes, the total number of reversals is $\sum_{j=1}^{m} j \cdot |L_j^I|$. If I has n bad nodes, there are at most n layers in the worst case (each layer contains one bad node). Thus, each node reverses at most n times. Since there are n bad nodes, the total number of reversals in the worst case is $O(n^2)$. Moreover, since a node reversal takes one time step and in the worst case all reversals are executed sequentially, the total number of reversals gives an upper bound on the stabilization time. Thus, we have the following.

COROLLARY 4.8 (work and time upper bounds for full reversal). *For any graph with an initial state with n bad nodes, the full reversal algorithm requires at most $O(n^2)$ work and time until stabilization.*

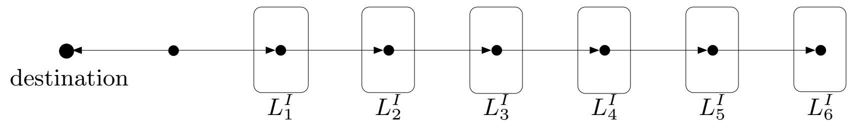

FIG. 3. *Worst-case work for full reversal: graph G_1 with $n = 6$ bad nodes.*

Actually, the upper bound of Corollary 4.8 is tight in both work and time in the worst case. First we show that the work bound is tight. Consider a graph G_1 which is an initial state with n layers of bad nodes such that each layer has exactly one node (see Figure 3 with $n = 6$). From Theorem 4.7, each node in the ith layer will reverse exactly i times. Thus, the sum of all the reversals performed by all the bad nodes is $n(n + 1)/2$, leading to the following corollary.

COROLLARY 4.9 (work lower bound for full reversal). *There is a graph with an initial state containing n bad nodes such that the full reversal algorithm requires $\Omega(n^2)$ work until stabilization.*

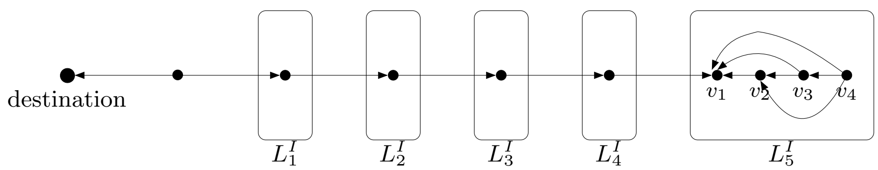

FIG. 4. *Worst-case stabilization time for full reversal: graph G_2 with $n = 8$ bad nodes, $m_1 = 5$ layers, and $m_2 = 4$ nodes in layer m_1.*

We will now show that the time bound of Corollary 4.8 is tight (within constant factors) in the worst case. Consider a graph G_2 (see Figure 4) with an initial state in which there are n bad nodes, such that it consists of $m_1 = \lfloor n/2 \rfloor + 1$ layers. The first $m_1 - 1$ layers contain one node each, while the last layer contains $m_2 = \lceil n/2 \rceil$ nodes. The last layer m_1 is as follows: there are m_2 nodes $v_1, v_2, \ldots, v_{m_2}$. Node v_i has outgoing links to all nodes v_j such that $j < i$. The node of layer $m_1 - 1$ has an outgoing link to node v_1 (see Figure 4).

From Theorem 4.7, we know that each node in layer m_1 requires exactly m_1 reversals before it becomes good. Since there are m_2 nodes in layer m_1, $m_1 \cdot m_2 = \Omega(n^2)$, reversals are required before these nodes become good. The key point is that any two nodes in layer m_1 are adjacent, so that all the reversals in that layer have to be performed sequentially. Thus, the reversals in layer m_1 alone take $\Omega(n^2)$ time, providing the following corollary.

COROLLARY 4.10 (time lower bound for full reversal). *There is a graph with an initial state containing n bad nodes such that the full reversal algorithm requires $\Omega(n^2)$ time until stabilization.*

Note that Corollary 4.10 subsumes Corollary 4.9, since a lower bound on time is also a lower bound on work.

5. Partial reversal algorithm. In this section, we present the analysis of the partial reversal algorithm. We first give an upper bound for work and stabilization time. We then present lower bounds for a class of worst-case graphs which is used to show that the upper bound is tight.

5.1. Upper bounds for partial reversal. Given an arbitrary initial state I, we give an upper bound on the work and stabilization time needed for the partial reversal algorithm. In order to obtain the bound, we decompose the bad nodes into levels and give an upper bound for the number of reversals of the nodes in each level; this then gives us an upper bound on work and time.

In particular, suppose that initial state I of the network contains n bad nodes. We say that a bad node v of state I is in *level* i if the shortest undirected path from v to a good node has length i. Note that the number of levels is no more than n.

The upper bound depends on the minimum and maximum heights of the nodes in state I. According to the partial reversal algorithm, each node v_i has a height (a_i, b_i, i). We will refer to a_i as the *alpha value* of node v_i. Let $a^{\max}$ and $a^{\min}$ denote the respective maximum and minimum alpha values of any node in the network in state I. Let $a^* = a^{\max} - a^{\min}$. We first give an upper bound on the alpha value of any node upon stabilization.

LEMMA 5.1. *After a node in level i becomes good its alpha value never exceeds $a^{\max} + i$.*

Proof. We prove the claim by induction on the number of levels. For the induction basis, consider a node v in level 1. If the alpha value of v becomes at least $a^{\max} + 1$, then v must have become a good node, since its height is more than the height of at least one adjacent node v' which is good in state I (from Lemma 3.2 v' does not reverse, and thus its alpha value remains at most $a^{\max}$). We need only show that during its final reversal, the alpha value of v will not exceed $a^{\max} + 1$. According to the partial reversal algorithm, the alpha value of v is equal to the smallest alpha value of its neighbors plus one. Moreover, the smallest alpha value of the neighbors cannot be greater than $a^{\max}$, since in I node v is adjacent to good nodes which don't reverse in future states (a consequence of Lemma 3.2). Thus, the alpha value of v will not exceed $a^{\max} + 1$ when v becomes a good node. Further, from Lemma 3.2, the alpha value of node v will not change thereafter.

For the induction hypothesis, let's assume that the alpha value of any node in level i, where $1 \leq i < k$, does not exceed $a^{\max} + i$, after that node becomes good. For the induction step, consider layer L_k. Let v be a node in level k. Clearly, node v is adjacent to some node in level $k-1$. From the induction hypothesis, the alpha value of every node in level $k-1$ cannot exceed $a^{\max} + (k-1)$ in any future state from I. If the alpha value of v becomes at least $a^{\max} + k$, then v must have become a good node, since its height is more than that of the adjacent nodes in level $k-1$ when these nodes become good. We need only show that during its final reversal, the alpha value of v will not exceed $a^{\max} + k$. According to the partial reversal algorithm, the alpha value of v is not more than the smallest alpha value of its neighbors plus one. Moreover, the smallest alpha value of the neighbors cannot exceed $a^{\max} + (k-1)$, which is the maximum alpha value of the nodes in level $k-1$ when these nodes become good. Thus, the alpha value of v will not exceed $a^{\max} + k$ when v becomes a good node. Further, from Lemma 3.2, the alpha value of node v will not change thereafter. ∎

At each reversal, the alpha value of a node increases by at least 1. Since the alpha value of a node can be as low as $a^{\min}$, Lemma 5.1 implies that a node in level i reverses at most $a^{\max} - a^{\min} + i$ times. Furthermore, since there are at most n levels, we obtain the following corollary.

COROLLARY 5.2. *A bad node will reverse at most $a^* + n$ times before it becomes a good node.*

Considering now all the n bad nodes together, Corollary 5.2 implies that the work needed until the network stabilizes is at most $n \cdot a^* + n^2$. Since in the worst case the reversal of the nodes may be sequential, the upper bound for work is also an upper bound for the time needed to stabilize. Thus we have the following.

THEOREM 5.3 (work and time upper bounds for partial reversal). *For any initial state with n bad nodes, the partial reversal algorithm requires at most $O(n \cdot a^* + n^2)$ work and time until the network stabilizes.*

5.2. Lower bounds for number of reversals. We will show that the upper bounds on work and time given in Theorem 5.3 are tight. We construct a class of worst-case graphs with initial states which require as much work and time as the upper bounds. In order to prove the lower bounds, we first determine how many reversals each node performs in the network.

In particular, consider a graph with an initial state I containing n bad nodes which can be decomposed into an even number m of layers $L_1, L_2, \ldots, L_{m-1}, L_m$ in the following way. A node is a *source* if all the links incident to the node are outgoing. The odd layers $L_1, L_3, \ldots, L_{m-1}$ contain only nodes which are nonsources, while the even layers $L_2, L_4, \ldots, L_m$ contain only nodes which are sources. The nodes in layer L_1 are the only bad nodes adjacent to good nodes. Let G denote the set of good nodes adjacent to layer L_1. Nodes in layer L_i may be adjacent only to nodes of the same layer and layers L_{i-1} and L_{i+1}.[3] We actually require that each node of L_i is adjacent to at least one node of L_{i-1} and at least one node of L_{i+1}. In addition, state I is taken so that all good nodes in the network have alpha value $a^{\max}$, while all the bad nodes have alpha value $a^{\min}$, where $a^{\max} > a^{\min}$. Let $a^* = a^{\max} - a^{\min}$. Instances of such an initial state are shown in Figures 5 and 6; at the end of this section we describe how to obtain such configurations with arbitrary large $a^{\max}$ in a mobile ad hoc network.

Given such an initial state I, we will give a lower bound on the number of reversals performed by each node at each layer until the network stabilizes. In order to obtain this result, we first show some necessary lemmas. A *full reversal* is a reversal in which a node reverses all of its links. Note that after a full reversal, a node becomes a source. We show that bad nodes which are sources always perform full reversals whenever they become sinks.

LEMMA 5.4. *Consider any state I_1 of the network in which a bad node v is a source with alpha value a. In a subsequent state I_2, in which node v becomes a sink for the first time after state I_1, the following occur:* (1) *v performs a full reversal, and* (2) *after the reversal of v, the alpha value of v becomes $a + 2$.*

Proof. In state I_1, since v is a source, all the adjacent nodes of v have alpha value at most a. Between states I_1 and I_2, each adjacent node of v has reversed at least once. We will show that in state I_2, the alpha value of each adjacent node of v is $a + 1$.

Let w be any adjacent node of v. First, we show that the alpha value of w in I_2 is at least $a + 1$. If in I_2 the alpha value of w is less than a, then v must have an outgoing link toward w, and thus v cannot possibly be a sink in I_2, a contradiction. Therefore, in I_2 the alpha value of w has to be at least a. Next, we show that this alpha value cannot be equal to a. If the alpha value of w in I_2 is a, then it must

[3] If $i = 1$, substitute G for L_{i-1}. If $i = m$, don't consider L_{i+1}.

be that the alpha value of w in I_1 was less than a (since w reversed between I_1 and I_2 and points toward v). When w was a sink the last time before I_2, w must have been adjacent to another node u with height $a-1$. When w reversed, its alpha value became a, but its incoming link from v didn't change direction since u had a smaller alpha value. Thus v cannot possibly be a sink in I_2, a contradiction. Therefore, the alpha value of w in I_2 cannot be equal to a, and it has to be at least $a+1$.

Next, we show that the alpha value of w cannot be greater than $a+1$. When w reverses, its alpha value is at most the minimum alpha value of its neighbors plus one. Therefore, since v is a neighbor of w with alpha value a, when w reverses, its alpha value cannot exceed $a+1$.

Therefore, the alpha value of w in state I_2 is exactly $a+1$. This implies that in I_2 all the neighbors of v have alpha value $a+1$. Thus, when v reverses, it performs a full reversal and its alpha value becomes $a+2$. ∎

Given state I described above, we give a lower bound for the alpha values of the nodes in each layer when the network stabilizes.

LEMMA 5.5. *When the network stabilizes from state I, the alpha values of all the nodes in layers L_{2i-1} and L_{2i}, $1 \leq i \leq m/2$, are at least $a^{\max}+i$.*

Proof. Let I' denote the state of the network when it stabilizes. We prove the claim by induction on i. For the basis case, where $i=1$, we consider layers L_1 and L_2. In state I, all the nodes of layer L_1 have only incoming links from G. In state I', there must exist a set S, consisting of nodes from L_1, such that the nodes in S have outgoing links pointing toward G.

Let v be a node in S. In state I', the alpha value of v is at least $a^{\max}$, since the nodes in G have alpha value $a^{\max}$. Actually, we will show that the alpha value of v in I' is larger than $a^{\max}$. Assume for contradiction that this value is $a^{\max}$. When node v reversed and obtained the alpha value $a^{\max}$, it cannot possibly have reversed its links toward G since, for these links, v adjusted only its second field on its height. Thus, in state I' node v is still bad, a contradiction. Therefore, in state I', node v has alpha value at least $a^{\max}+1$; thus, in state I', all nodes in set S have alpha value at least $a^{\max}+1$.

Now, consider the rest of the nodes in layers L_j, $j \geq 1$. Let w be any such node. In state I', w is good, and thus there exists a directed path from w to a good node in G. This path has to go through the nodes of S; thus each node in the path must have alpha value at least $a^{\max}+1$, which implies that w has alpha value at least $a^{\max}+1$. Therefore, in state I', all nodes in L_1 and L_2 (including S) have alpha value at least $a^{\max}+1$.

Now, let's assume that the claim holds for all $1 \leq i < k$. We will show that the claim is true for $i=k$. We consider layers L_{2k-1} and L_{2k}. In state I all the nodes of layer L_{2k-1} have only incoming links from L_{2k-2}. In state I', there must exist a set S, consisting of nodes of L_{2k-1}, such that the nodes in S have outgoing links pointing toward L_{2k-2}. The rest of the proof is similar to the induction basis, where now we show that the nodes in S in state I', have alpha values at least $a^{\max}+k$, which implies that all nodes in L_{2k-1} and L_{2k} have alpha value at least $a^{\max}+k$. ∎

We are now ready to show a central theorem for the lower bound analysis, which is a lower bound on the number of reversals for the nodes of each layer. This result will help us to obtain lower bounds for work and time needed for stabilization.

THEOREM 5.6 (lower bound on reversals for partial reversal). *Until the network stabilizes, each node in layers L_{2i-1} and L_{2i}, $1 \leq i \leq m/2$, will reverse at least $\lfloor (a^*+i)/2 \rfloor$ times.*

Proof. Consider a bad node v of L_{2i}. Node v is a source in state I. Lemma 5.4 implies that whenever v reverses in the future, it reverses all of its incident links, and therefore it remains a source. Moreover, Lemma 5.4 implies that every time that v reverses, its alpha value increases by 2. Lemma 5.5 implies that when the network stabilizes, the alpha value of v is at least $a^{\max} + i$. Since in state I the alpha value of v is $a^{\min}$, node v reverses at least $\lfloor (a^* + i)/2 \rfloor$ times after state I. Similarly, any other node in L_{2i} reverses at least $\lfloor (a^* + i)/2 \rfloor$ times.

Consider now a bad node w of L_{2i-1}. Node w is adjacent to at least one node u in layer L_{2i}. In state I, node u is a source, and it remains a source every time that u reverses (Lemma 5.4). Since u and w are adjacent, the reversals of u and w should alternate. This implies that node w reverses at least $\lfloor (a^* + i)/2 \rfloor$ times, since node u reverses at least $\lfloor (a^* + i)/2 \rfloor$ times. Similarly, any other node in L_{2i-1} reverses at least $\lfloor (a^* + i)/2 \rfloor$ times. ∎

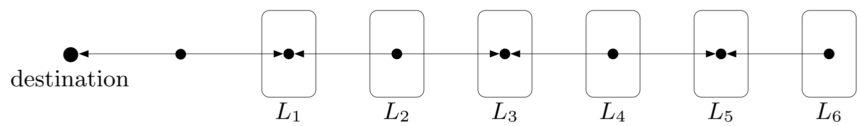

FIG. 5. *Worst-case work for partial reversal: graph G_3 with $n = 6$ bad nodes.*

Using Theorem 5.6 we now give worst-case graphs for work and stabilization time, which show that the upper bounds of Theorem 5.3 are tight. First, we give the lower bound on work. Consider a graph G_3 which is in state I, as described above, in which there are n bad nodes, where n is even, and there is exactly one bad node in each layer (see Figure 5). From Theorem 5.6, each node in the ith layer will reverse at least $\lfloor (a^* + \lceil i/2 \rceil)/2 \rfloor$ times before the network stabilizes. Thus, the sum of all the reversals performed by all the bad nodes is at least $\sum_{i=1}^{n} \lfloor (a^* + \lceil i/2 \rceil)/2 \rfloor$, which is $\Omega(n \cdot a^* + n^2)$. Thus, we have the following corollary.

COROLLARY 5.7 (work lower bound for partial reversal). *There is a graph with an initial state containing n bad nodes such that the partial reversal algorithm requires $\Omega(n \cdot a^* + n^2)$ work until stabilization.*

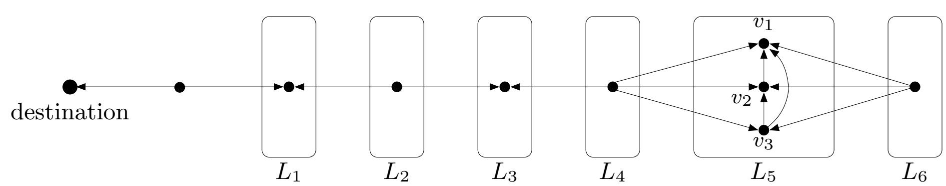

FIG. 6. *Worst-case stabilization time for partial reversal: graph G_4 with $n = 8$ bad nodes, $m_1 = 6$ layers, and $m_2 = 3$ nodes in layer $m_1 - 1$.*

Now we give the lower bound on time. Consider a graph G_4 in a state I as described above, in which there are n bad nodes, where $n/2$ is even. The graph consists of $m_1 = n/2 + 2$ layers. The first $m_1 - 2$ layers contain one node each, while layer $m_1 - 1$ contains $m_2 = n/2 - 1$ nodes, and layer m_1 contains 1 node. The layer $m_1 - 1$ is as follows: there are m_2 nodes $v_1, v_2, \ldots, v_{m_2}$. Node v_i has outgoing links to all nodes v_j such that $j < i$ (see Figure 6). Note that each node in layer v_i is connected to the nodes in the adjacent layers from the specification of state I.

From Theorem 5.6, we know that each node in layer $m_1 - 1$ requires at least $k_1 = \lfloor (a^* + \lceil (m_1 - 1)/2 \rceil)/2 \rfloor$ reversals before it becomes a good node. Since layer

$m_1 - 1$ contains m_2 nodes, at least $k_1 \cdot m_2 = \Omega(n \cdot a^* + n^2)$ reversals are required before these bad nodes become good nodes. All these reversals have to be performed sequentially, since the nodes of layer $m_1 - 1$ are adjacent, and any two of these nodes cannot be sinks simultaneously. Thus, we have the following corollary.

COROLLARY 5.8 (time lower bound for partial reversal). *There is a graph with an initial state containing* n *bad nodes such that the partial reversal algorithm requires* $\Omega(n \cdot a^* + n^2)$ *time until stabilization.*

Note that Corollary 5.8 subsumes Corollary 5.7, since a lower bound on time is also a lower bound on work.

We now describe scenarios in mobile ad hoc networks which could result in the state I of graph G_4 and with arbitrary a^* value and number of nodes n. We first describe how to obtain an arbitrary $a^{\max}$ value in a small graph. Consider a graph consisting of the destination node and two nodes w_1 and w_2. Initially node w_1 points only to w_2, which points to the destination; further, the alpha values of the nodes are all zero. Next, w_1 moves and gets also connected to the destination, without changing its height. Now w_2 moves and gets disconnected from the destination, but it still is connected to w_1. However, w_2 is now a sink, and thus it performs a reversal, where its alpha value increases by one, since it has one neighbor (w_1). This scenario can be repeated an arbitrary number of times with the roles of w_1 and w_2 interchanged. This results in a state with an arbitrary value of $a^{\max}$.

Next, we describe how to obtain arbitrary $a^* = d$ in a state I of graph G_4. Let $a^*(I)$ denote a^* in state I. Consider a graph H in an initial state I' in which all the nodes are good and all have alpha value equal to zero (thus, $a^*(I') = 0$). The nodes, except the destination, are divided into three components H_1, H_2, and H_3. Graph H_1 consists of n nodes and is in a state isomorphic to the bad nodes in graph G_4. Graph H_2 is a set of good nodes such that each node of H_1 is connected with an outgoing link to a good node in H_2; essentially, the nodes of H_1 are good because they are connected to the nodes of H_2. Graph H_3 is a network consisting only of nodes w_1 and w_2 as described in the previous paragraph. From state I' we obtain a state I'' as follows. We let the nodes in H_3 oscillate (as described in the previous paragraph) until the alpha value of w_1 or w_2 is equal to d. Suppose that w_1 is the node that gets height d first and we stop the oscillation immediately when w_1 gets connected directly to the destination. Note that the nodes in H_1 and H_2 haven't changed their heights since I', and therefore their alpha values remain zero in I'' (thus, $a^*(I'') = d$). Now, from state I'' we will obtain a state I as follows. The nodes in H_2 and the node w_2 disappear from the network; also, the node in the first layer of H_1 gets connected to w_1. The resulting network configuration and state I are the same as in graph G_4. Since the nodes in H_1 haven't changed their original alpha value (zero) and the node w_1 has the largest alpha value d, we obtain $a^*(I) = d$, as needed.[4]

6. Deterministic algorithms.

In this section, we give worst-case lower bounds for the work and time needed for stabilization for any deterministic link reversal algorithm. Given an arbitrary deterministic function g, we will establish the existence of a family of graphs with initial states containing any number of $n > 2$ bad nodes which require at least $\Omega(n^2)$ work and time until stabilization. The lower bounds follow from a lower bound on the number of reversals performed by each node, which we describe next.

[4] A similar scenario could give a state I with arbitrary a^*, where $a^{\min}$ is larger than zero. However, the state we gave with $a^{\min} = 0$ suffices for our lower bounds.

6.1. Lower bound on number of reversals. Given a graph G, we construct a state I in which we determine a lower bound on the number of reversals required by each bad node until stabilization. In particular, we decompose the bad nodes into levels and show that the node at level i reverses at least $i-1$ times until stabilization. We then use this result to obtain lower bounds for work and time.

The construction of state I depends on a level decomposition of the network. Let s be the destination node. A node v (bad or not) is defined to be in *level* i if the (undirected graph) distance between v and s is i. Thus s is in level 0, and a node in level i is connected only with nodes in levels $i-1$, i, and $i+1$. Let m denote the maximum level of any node.

We now construct recursively states $I_{m+1}, I_m, I_{m-1}, \dots, I_2$ such that state $I = I_2$. The basis of the recursion is state I_{m+1}. The construction is as follows.

- In state I_{m+1} every node is good. Further, the heights of nodes in levels $1, 2, 3, \dots, m$ are in increasing order with the levels, i.e., given two vertices u, v at respective levels l_u, l_v with $l_u < l_v$, u's height is less than v's. An example assignment of heights in state I_{m+1} is to set a node's height to be equal to its level.
- Suppose we have constructed state I_i, where $m+1 \geq i > 2$. We construct state I_{i-1} as follows:
  - For every node in levels $i-1, i, \dots, m$, the height of the node in I_{i-1} is the same as its height in I_i.
  - Let $\max_i$ denote the maximum height of a node in the destination oriented graph that is reached by an execution starting from I_i. (We note that from Corollary 3.5, $\max_i$ does not depend on the actual execution sequence, but only on the initial state I_i.) For every node v in levels $1, 2, \dots, i-2$, v's height in I_{i-1} is assigned to be v's height in I_i plus $\max_i$.

In the above construction, we assumed that the function g converges at a finite amount of time to a stable state starting from any initial state I_i. This assumption doesn't hurt the generality of our analysis, since if g didn't stabilize it would trivially require at least $\Omega(n^2)$ work and time, for n bad nodes, and thus our main result still holds. So, without loss of generality, we will assume that g stabilizes.

Next, we show that each state I_i, $m+1 \geq i \geq 2$, satisfies the following properties:

$P1_i$: The heights of nodes in levels $1, 2, 3, \dots, i-1$ are in increasing order with the levels; i.e., given two vertices u, v at respective levels l_u, l_v with $l_u < l_v$, u's height is less than v's. Thus, every node in levels $1, 2, \dots, i-1$ is a good node.

$P2_i$: The heights of nodes in levels $i-1, i, i+1, \dots, m$ are in decreasing order with the levels, i.e., given two vertices u, v at respective levels l_u, l_v with $l_u < l_v$, u's height is greater than v's. Thus, every node in levels $i, i+1, \dots, m$ is bad. In the case when $i = m+1$, no node is bad.

$P3_i$: Starting from initial state I_i, every node in level j, $j = i, i+1, \dots, m$, reverses at least $j-i+1$ times until stabilization. In the case when $i = m+1$, no node reverses.

For $i = m+1, \dots, 2$, we will now argue about the number of reversals starting from state I_i until stabilization. From Corollary 3.5, we know that all executions of a deterministic algorithm starting from the same initial state are essentially identical. In particular, the number of reversals of each node in every execution is the same.

For convenience, we consider a specific execution E_i which starts from initial state I_i and reverses nodes in the following order: next reverse the bad node which has the smallest height in the current state. Clearly, such a node is a sink, and hence a candidate for reversal. The number of reversals of a node in any execution starting from I_i is equal to the number of reversals of the node in E_i.

LEMMA 6.1. *State I_i, $m+1 \geq i \geq 2$, satisfies properties $P1_i$, $P2_i$, and $P3_i$.*

Proof. The proof is by induction on i. For the induction basis, state I_{m+1} clearly satisfies all the properties $P1_{m+1}$, $P2_{m+1}$, $P3_{m+1}$ from the construction of this state. Suppose now that state I_i, where $m+1 \geq i > 2$, satisfies the respective three properties. We will show that state I_{i-1} satisfies the respective properties too. It can be easily checked that properties $P1_{i-1}$ and $P2_{i-1}$ are satisfied in I_{i-1}. We focus on property $P3_{i-1}$.

An execution is a sequence of reversals. We say that two reversals (v, h, H) and (v, h', H') of the same node in different network states are *equal* if the heights of the node and its neighbors are the same in both states, namely, $h = h$ and $H = H'$. If two reversals are equal, then the heights of the node after the reversals are the same, since we are using a deterministic height increase function g. An execution E is said to be a prefix of an execution E' if the reversal sequence constituting E is elementwise equal to a prefix of the reversal sequence constituting E'. In Lemma 6.2, we show that E_i is a prefix of E_{i-1}.

Consider the state of the system which started in I_{i-1}, but after executing the reversals in E_i. In this state, the height of each node in levels $1, 2, \ldots, i-2$ is greater than $\max_i$ by construction (it was greater than $\max_i$ in I_{i-1}, and heights can never decrease). Let v be a node in level $l_v > i-2$. After the execution segment E_i, v's height is the same as the final height in the destination oriented graph reached from I_i, and by the definition of $\max_i$, this is no more than $\max_i$.

Thus, in the current state, the height of every node in levels $i-1, i, \ldots, m$ is less than the height of every node in levels $1, 2, \ldots, i-2$. Consider any node u in level $l_u \geq i-1$. In the final destination oriented graph, there is a path of decreasing height from u to the destination s, and this path contains at least one node from levels $1, \ldots, i-2$. Thus, in the final state, u's height is greater than the height of some node in levels $1, \ldots, i-2$, while it was less to begin with. This implies that u must have reversed at least once until stabilization.

In E_i, each node in level j, $j = i, i+1, \ldots, m$ has reversed at least $j-i+1$ times. If we add an extra reversal to all nodes in levels $i-1, i, \ldots, m$, then in E_{i-1}, each node in levels $j = i-1, i, \ldots, m$ reverses at least $j-i+2$ times, thus proving property $P3_{i-1}$. ∎

LEMMA 6.2. *Execution E_i is a prefix of execution E_{i-1}.*

Proof. Executions E_i and E_{i-1} start from states I_i and I_{i-1}, respectively. Let $E_i = r_1^i, r_2^i, \ldots, r_f^i$ and $E_{i-1} = r_1^{i-1}, r_2^{i-1}, \ldots$. We prove by induction that $r_j^i = r_j^{i-1}$ for $j = 1, \ldots, f$.

Base case. The nodes with the lowest height in I_i and I_{i-1} are the same node v, and v lies in layer m. The heights of all nodes in layer $m-1$ are the same in I_i and I_{i-1} by construction. Thus, all of v's neighbors have the same height in I_i and I_{i-1}, so that $r_1^i = r_1^{i-1}$.

Inductive case. Suppose that $r_1^i, r_2^i, \ldots, r_l^i$ is identical to $r_1^{i-1}, r_2^{i-1}, \ldots, r_l^{i-1}$ for some $l < f$. Let I_l^i and I_l^{i-1}, respectively, denote the state of the system starting from I_i after reversals $r_1^i, r_2^i, \ldots, r_l^i$, and the state of the system starting from I_{i-1} after reversals $r_1^{i-1}, r_2^{i-1}, \ldots, r_l^{i-1}$.

Let v be the bad node with the lowest height in I_l^i so that r_{l+1}^i reverses v. We claim that this is also the bad node with the lowest height in I_l^{i-1}. The reason is as follows.

Node v must be at a level $l_v \geq i$, since E_i does not reverse any nodes at a lower level than i. All nodes in levels $i-1$ or greater have the same heights in I_l^i and I_l^{i-1}, due to the induction hypothesis, and these are all less than $\max_i$. All nodes in levels $i-2$ or less in I_l^{i-1} have heights greater than $\max_i$ by construction. Thus, the bad node with the minimum height in I_l^{i-1} is also v, and its neighbors also have the same heights as in I_l^i, implying that r_{l+1}^i is the same as r_{l+1}^{i-1}. This completes the proof. ∎

We are now ready to show the main result of this section.

THEOREM 6.3 (lower bound on reversals for deterministic algorithms). *Given any graph G and any height increase function g, there exists an initial state I (an assignment of heights to the nodes of G) which causes each node in level $i > 0$ to reverse at least $i-1$ times until stabilization.*

Proof. Let m denote the maximum node level. We first construct a sequence of initial states $I_{m+1}, I_m, \ldots, I_2$ as described above. Lemma 6.1 implies that starting from initial state I_i, each node in level $j, j \geq i$ reverses at least $j-i+1$ times until stabilization (property $P3_i$). We take $I = I_2$. ∎

6.2. Worst-case graphs. Here we give lower bounds on the work and time for any deterministic algorithm. Theorem 6.3 applies to any graph. Consider the list graph G_1 with $n+2$ nodes, shown in Figure 3 and described in section 4.3. We construct a state I with n bad nodes as described in section 6.1. From Theorem 6.3, the lower bound for the worst-case number of reversals of any reversal algorithm on state I is the sum of the reversals of each bad node: $1+2+\cdots+n = \Omega(n^2)$. Thus we have the following corollary.

COROLLARY 6.4 (work lower bound for deterministic algorithms). *There is a graph with an initial state containing n bad nodes such that any deterministic reversal algorithm requires $\Omega(n^2)$ work until stabilization.*

We can derive a similar lower bound on the time needed for stabilization. We use the graph G_4 with $n+2$ nodes, shown in Figure 4. The structure of the graph, and the parameters m_1 and m_2, are defined as in section 5.2 with respect to $n+1$. We construct a state I with n bad nodes as described in section 6.1. From Theorem 6.3, we know that each node in level $m_1 - 1$ of G_4 requires at least $(m_1 - 2)$ reversals before it becomes a good node. Level $m_1 - 1$ contains m_2 nodes. Therefore, at least $(m_1 - 2) \cdot m_2 = \Omega(n^2)$ reversals are required before these nodes become good nodes. All these reversals have to be performed sequentially, since the nodes of layer $m_1 - 1$ are adjacent, and no two of these nodes can be sinks simultaneously. Thus, we have the following corollary.

COROLLARY 6.5 (time lower bound for deterministic algorithms). *There is a graph with an initial state containing n bad nodes such that any deterministic reversal algorithm requires $\Omega(n^2)$ time until stabilization.*

7. Conclusions and discussion. We presented a worst-case analysis of link reversal routing algorithms in terms of work and time. We showed that for n bad nodes, the GB full reversal algorithm requires $O(n^2)$ work and time, while the partial reversal algorithm requires $O(n \cdot a^* + n^2)$ work and time. The above bounds are tight in the worst case. Our analysis for the full reversal is exact. For any network, we present a decomposition of the bad nodes in the initial state into *layers*, which allows us to predict *exactly* the work performed by each node in *any* distributed execution.

Furthermore, we show that for any deterministic reversal algorithm on a given graph, there exists an assignment of heights to all the bad nodes in the graph such that if a bad node d hops away from its closest good node, then it has to reverse d times before stabilization. Using this, we show that there exist networks and initial states with n bad nodes such that the algorithm needs $\Omega(n^2)$ work and time until stabilization. As a consequence, from the worst-case perspective, the full reversal algorithm is work and time optimal, while the partial reversal algorithm is not. Since a^* can grow arbitrarily large, the full reversal algorithm outperforms the partial reversal algorithm in the worst case.

Since it is known that partial reversal performs better than full reversal in some cases, it would be interesting to find a variation of the partial reversal algorithm, which is as good as full reversal in the worst case. Another research problem is to analyze the average performance of link reversal algorithms. It would be also interesting to extend our analysis to nondeterministic algorithms, such as randomized algorithms, in which the new height of a sink is some randomized function of the neighbors' heights.

Acknowledgments. We thank the reviewers for their valuable comments and suggestions. We also thank Srikanth Surapaneni for helping in the preparation of an earlier version of this paper.

REFERENCES

[1] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva, *A performance comparison of multi-hop wireless ad hoc network routing protocols*, in Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM), 1998, ACM, New York, pp. 85–97.

[2] C. Busch, S. Surapaneni, and S. Tirthapura, *Analysis of link reversal routing algorithms for mobile ad hoc networks*, in Proceedings of the Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), 2003, pp. 210–219.

[3] I. Chatzigiannakis, E. Kaltsa, and S. Nikoletseas, *On the effect of user mobility and density on the performance of routing protocols for ad-hoc mobile networks*, Wireless Communication and Mobile Computing, 4 (2004), pp. 609–621.

[4] I. Chatzigiannakis, S. Nikoletseas, and P. Spirakis, *Distributed communication algorithms for ad-hoc mobile networks*, J. Parallel Distrib. Comput., 63 (2003), pp. 58–74.

[5] M. S. Corson and A. Ephremides, *A distributed routing algorithm for mobile radio networks*, in Proceedings of the IEEE Military Communications Conference (MILCOM), 1989, pp. 210–213.

[6] M. S. Corson and A. Ephremides, *A distributed routing algorithm for mobile wireless networks*, ACM/Baltzer Wireless Networks J., 1 (1995), pp. 61–82.

[7] E. M. Gafni and D. P. Bertsekas, *Distributed algorithms for generating loop-free routes in networks with frequently changing topology*, IEEE Trans. Comm., 29 (1981), pp. 11–18.

[8] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, *Directed diffusion for wireless sensor networking*, IEEE/ACM Trans. Networking (TON), 11 (2003), pp. 2–16.

[9] N. Malpani, J. L. Welch, and N. Vaidya, *Leader election algorithms for mobile ad hoc networks*, in Proceedings of the Fourth Annual ACM International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communication (DIAL-M), 2000, pp. 96–103.

[10] V. D. Park and M. S. Corson, *A highly adaptive distributed routing algorithm for mobile wireless networks*, in Proceedings of the Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), 1997, pp. 1405–1413.

[11] V. S. Park and M. S. Corson, *A performance comparison of the temporally-ordered routing algorithm and ideal link-state routing*, in Proceedings of the Third Annual IEEE International Symposium on Computers and Communications, 1998, pp. 592–598.

[12] C. E. Perkins, *Ad Hoc Networking*, Addison–Wesley, Reading, MA, 2000.

[13] R. Rajaraman, *Topology control and routing in ad hoc networks: A survey*, SIGACT News, 33 (2002), pp. 60–73.

[14] S. R. Das, R. Castaneda, Y. Jiangtao, and R. Sengupta, *Comparative performance evaluation of routing protocols for mobile, ad hoc networks*, in Proceedings of the Seventh Annual IEEE International Conference on Computer Communications and Networks (IC3N), 1998, pp. 153–161.