



ACADEMIC
PRESS

Available online at www.sciencedirect.com

SCIENCE @ DIRECT®

J. Parallel Distrib. Comput. 63 (2003) 87–96

Journal of
Parallel and
Distributed
Computing

<http://www.elsevier.com/locate/jpdc>

Self-stabilizing multicast protocols for ad hoc networks[☆]

Sandeep K.S. Gupta^{a,1} and Pradip K Srimani^{b,*}

^aDepartment of Computer Science, Colorado State University, Ft. Collins, CO 80523, USA

^bDepartment of Computer Science, Clemson University, Clemson, SC 29634-0974, USA

Received 17 October 2002

Abstract

We propose two distributed algorithms to maintain respectively the minimum weight spanning tree (MST) based multi-cast tree and the shortest path (SPST) multi-cast tree in a given ad hoc network for a given multi-cast group; our algorithms are fault tolerant (reliable) in the sense that the algorithms can detect occasional link failures and/or new link creations in the network (due to mobility of the hosts) and can readjust the multi-cast tree. Our approach is to use the paradigm of self-stabilization in distributed fault tolerance. We provide time complexity analysis of the algorithms in terms of the number of rounds needed for the algorithm to stabilize after a topology change, where a round is defined as a period of time in which each node in the system receives beacon messages from all its neighbors. In any ad hoc network, the participating nodes periodically transmit beacon messages for message transmission as well as to maintain the knowledge of the local topology at the node; as a result the nodes get the information about its neighbor nodes synchronously (at specific time intervals). Thus, the paradigm to analyze the complexity of the self-stabilizing algorithms in the context of ad hoc networks is very different from the traditional concept of adversary oracle used in proving the convergence and correctness of self-stabilizing distributed algorithms in general.

© 2002 Published by Elsevier Science (USA).

Keywords: Self-stabilizing protocol; Distributed system; Multi-cast protocol; Fault tolerance; Convergence; System graph

1. Introduction

Mobile ad hoc networks are being increasingly used for military operations, law enforcement, rescue missions, virtual class rooms, and local area networks [Sto02]. A mobile multi-hop network consists of n identical mobile hosts (nodes) with unique identifiers (ids) $1, \dots, n$ (node ids are always unique while the assignment of natural numbers is done for ease of explanation only). These mobile hosts communicate among each other via message broadcast. When a node transmits (broadcasts) a message, the nodes in the *coverage area* of the sender can simultaneously receive the message. A node i is called a *neighbor* of node j in the network if node j is in the *coverage area* of node i . This relationship is time varying since the nodes can and do

move. At any given time a node i can correctly receive a message from one of its neighbors, say j , iff j is the only neighbor of i transmitting at that time. A multiple access protocol is used by a nodes to get contention-free access to the wireless channel. Since, a limited communication bandwidth is shared between several neighboring nodes the available communication bandwidth between any two neighboring nodes depends upon number of nodes in the vicinity and their communication activity. Further, since nodes are battery-powered they can go to sleep in order to conserve battery power [HDL02]. Hence, in mobile ad hoc network (i) old links fail and new links are formed over time due to node movement (ii) nodes can be unreachable since they may go to sleep (iii) the bandwidth available on a (logical) link between two neighboring nodes varies with time, and (iv) the topology may get disconnected.

Multi-cast is an important group communication primitive; it involves sending a message to an arbitrary subset of nodes in the network. It is useful in a large number of applications in a distributed system, where a set of nodes need to be informed of a specific event.

[☆]This research was supported by NSF Awards # ANI-0073409 and ANI-0196156.

*Corresponding author.

E-mail address: srimani@cs.clemson.edu (P.K. Srimani).

¹Present address: Department of Computer Science and Engineering, Arizona State University, Tempe, AZ 85287-5406, USA.

Multi-cast protocols for distributed systems with a static network have been widely studied. Multi-cast protocols are evaluated in terms of (1) bandwidth consumed in the entire process and (2) the maximum delay incurred in delivering message to any member of the multi-cast group. A multi-cast message can be sent to each member of the multi-cast group separately; this wastes bandwidth since the message may be repeatedly sent over the same communication link. Conceptually, this wastage can be avoided by constructing a minimal cost tree spanning only the members of the multi-cast group in the entire network graph and then forwarding the message from the root of the tree (source of the message) along the tree edges. Unfortunately, the problem of computing the minimal cost multi-cast tree spanning an arbitrary number of nodes (members of the multi-cast group) in a given weighted graph is NP-complete [Kar72]; hence, people have concentrated on designing heuristic algorithms for designing near optimal multi-cast trees for given multi-cast groups and analyzing their performance.

Our purpose in the present paper is to propose two distributed algorithms to maintain respectively the minimum weight spanning tree (MST) based multi-cast tree and the shortest path (SPST) multi-cast tree in a given ad hoc network for a given multi-cast group; our algorithms are fault tolerant (reliable) in the sense that the algorithms can detect occasional link failures and/or new link creations in the network (due to mobility of the hosts) and can readjust the multi-cast tree. Our approach is to use the relatively new paradigm for distributed fault tolerance, e.g., *self-stabilization* to design the fault tolerant distributed algorithms. Conceptually, the algorithms involves three logical steps: (1) construction of the minimal spanning tree (MST) or the shortest-path spanning tree (SPST) of the mobile network graph in presence of topology change due to node mobility; (2) once the MST/SPST stabilize for the new topology, rooting the MST/SPST with respect to the given source node of the multi-cast message (establishing unique parent pointer for each node in the MST/SPST); pruning from the MST/SPST the nodes that are not needed to send the message to the multi-cast group members. The computation is performed in a distributed manner by using the mechanism of beacon messages. Mobile ad hoc networks use periodic beacon messages (also called “keep alive” messages) to inform their neighbors of their continued presence. A node presumes that a neighboring node has moved away unless it receives its beacon message at stipulated interval. This beaconing message provides an inexpensive way of periodically exchanging additional information between neighboring nodes in the network. In our algorithm, a nodes takes action after receiving beacon messages (along with algorithm-related information) from all the neighboring nodes. The protocols use

some concepts from an existing self-stabilizing distributed algorithm [CS94] and the preliminary versions of the protocols were presented in [GS99,GBS00]. The most important contribution of the paper involves the analysis of the time complexity of the algorithms in terms of the number of rounds needed for the algorithm to stabilize after a topology change, where a round is defined as a period of time in which each node in the system receives beacon messages from all its neighbors. In any ad hoc network, the participating nodes always periodically transmit beacon messages for message transmission as well as to maintain the dynamic topology; as a result the nodes get the information about its neighbor nodes synchronously (at specific time intervals). Thus, the paradigm to analyze the complexity of the self-stabilizing algorithms in the context of ad hoc networks is very different from the traditional concept of adversarial oracle used in proving the convergence and correctness of self-stabilizing distributed algorithms in general.

2. System model

2.1. Assumptions

We make the following assumptions about the system.

- A data link layer protocol at each node i maintains the identities of its neighbors in some list $neighbors(i)$. This data link protocol also resolves any contention for the shared medium by supporting logical links between neighbors and ensures that a message sent over a correct (or functioning) logical link is correctly received by the node at the other end of that link. The logical links between two neighboring nodes are assumed to FIFO, i.e., messages are received in the same order that they are sent. The link layer protocol informs the upper layer of any creation/deletion of logical links using the *neighbor discovery protocol* [Per01] described below.
- Each node periodically (at intervals of t_b) broadcasts a *beacon* message. This forms the basis of *neighbor discovery protocol* (details can be found in [Sto02]. At the logical level, when a node i receives the beacon signal from a node j which is not in its neighbors list $neighbors(i)$, it adds j to its neighbors list (data structure $neighbors_i$ at node i), thus establishing link (i,j) . For each link (i,j) , node i maintains a timer t_{ij} for each of its neighbors j . If node i does not receive a beacon signal from neighbor j in time t_b , it assumes that link (i,j) is no longer available and removes j from its neighbor set. Upon receiving a beacon signal from neighbor j , node i resets its appropriate timer.

- When a node j sends a beacon message to any of its neighbors, say node i , it includes some additional information in the message that are used by node i to compute the cost of the link (i, j) as well as other information regarding the state of the node j , as used in the algorithm.
- The topology of the ad hoc network is modeled by a (undirected) graph $G = (V, E)$, where V is the set of nodes and E is the set of links between neighboring nodes. We assume that the links between two adjacent nodes are always bidirectional. Since the nodes are mobile, the network topology changes with time. We assume that no node leaves the system and no new node joins the system; we also assume that transient link failures are handled by the link layer protocol by using timeouts, retransmissions, and per hop acknowledgments. Thus, the network graph has always the same node set but different edge sets. Further, we assume that the network topology remains connected. These assumptions hold in mobile ad hoc networks in which the movement of nodes is co-ordinated to ensure that the topology does not get disconnected.
- There is an underlying unicast routing protocol to send unicast messages between two arbitrary nodes in the network.

2.2. Implications on protocol execution

The convergence and correctness of self-stabilizing distributed algorithms in general are proved by showing that the protocol converges to a legitimate state in a finite amount of time [Sch93]; the proof uses an adversarial oracle and does not assume any restriction on how many privileged nodes that are eligible to make a correction of their respective states based on local knowledge) make a move at any given point of time and the nodes may act in a way to make the convergence time the longest possible. In case of a mobile ad hoc network in our system model, we can provide a tighter analysis of the convergence time of the proposed protocol.

Each node periodically broadcasts a beacon message to its neighbors and this period is same for each node in the system. Let us define a *round* of computation as the time between two consecutive beacon message broadcast (i.e. the period of beacon message broadcast). Thus, in each round, every node that is privileged due to the actions taken by the nodes during the immediate past round, will make a move to make the node locally stable. Let D denote the diameter of the underlying network in terms of the number of edges traversed (as opposed to the distance of the path traversed); this diameter does not reflect the actual costs of the edges in the system graph. Let m_1 and m_2 denote respectively the minimum and the maximum edge weight in the system graph.

Remark 1. The ratio $\lceil \frac{m_2}{m_1} \rceil$ plays a very important role in determining the convergence time of the protocol. Note that, in our correctness proof of the protocol, we have not made any assumption about the possible values of this ratio. In an adversarial oracle this ratio can be very large (ratio of the largest real number to the smallest positive non zero real number that can be stored), while in the context of an ad hoc network, the ratio (range of link costs) would be small. For example in “revised ARPANET routing metric” the most expensive link is only seven times the cost of the least expensive link [KZ89]. The reason being that there is a relationship link cost and link utilization. A link which has a very low cost gets overly utilized since it is included in many shortest path whereas a link with a very high cost has an extremely low utilization since it hardly gets included in any path. Analysis of Internet packet traces show that, if the range of link cost is very wide, say 1–127, a high percentage of network traffic is destined for a small number of network links [PD96]. This results in overall very poor utilization of the network. In mobile ad hoc network this can aggravate the problem of bandwidth scarcity even further. Hence, in mobile ad hoc networks the ratio of link of most expensive to least expensive link is expected to be very small [Kes97].

3. Shortest path tree (SPST) protocol

Let $G = (V, E)$ be a symmetric graph (with no self-loops) representing the ad hoc mobile network where V is the set of nodes (mobile hosts), $|V| = n$, and E is the set of edges. Without loss of generality we assume that the nodes are numbered 1 through n . Every edge $e_{ij} \in E$ has a *positive (non-zero)* weight w_{ij} assigned to it (note that $w_{ij} = w_{ji}$ for all i and j since the graph is symmetric). Let $Adj(i)$ represent the set of all nodes adjacent to node i . Let r be the designated node that serves as a *center node* for multi-cast tree. The center node is called as the core node in core-based tree (CBT) protocol [BFC93] and as a rendezvous point in protocol-independent multi-cast (PIM) protocol [DEF96]. A center-based multi-cast tree is shared by all the multi-cast group members. Whenever a source member in the multi-cast group wants to send a message to all the group members it sends the message to the center node which initiates the distribution of the message down the multi-cast tree rooted at it. In this paper we are interested in maintaining a shortest-path tree rooted at the center node. In static networks there exist several protocols for determining good center nodes [BFC93, DEF96]. Determination of a good center node in mobile ad hoc network is a research topic in itself and is out of the scope of this paper.

We want to design a self-stabilizing distributed algorithm that maintains a shortest path spanning tree (SPST) of the given graph that is rooted at the given node r (the shortest path spanning tree of a given symmetric graph is defined to be a spanning tree of the graph where the distance of each node from the given root in the tree is equal to the shortest distance of the node from the root node in the original graph). The knowledge of the SPST need not be duplicated at each node; each node will only know its unique predecessor in the said SPST. Each node attempts to compute the length (cost) of the shortest path to the given reference node r . Let $S_i(r)$ denotes the length of the shortest path from node i to node r .

Remark 2. For all i , $S_i(r)$ is determined by the topology of the graph and the weights assigned to the edges and that $S_r(r) = 0$ (no self-loops).

3.1. The algorithm

Each node $i \in V$ maintains a local variable $D_i(r)$; $D_i(r)$ is the current estimate of $S_i(r)$ known at node i and it determines the *local state* of node i . In addition, each node i also maintains a predecessor pointer P_i pointing to one of the nodes in $Adj(i)$; P_i points to the node adjacent to node i in the currently estimated shortest path from node i to node r .

Remark 3. In an illegitimate state, any node i (including the reference node r) can have an arbitrary value for $D_i(r)$ between 0 and some large positive number which we shall call MAXINT (determined by the length of the registers holding these variables). Similarly, in an illegitimate state, any node can point to an arbitrary node to be its immediate predecessor in the shortest path spanning tree.

We introduce the following set $\mathcal{N}(i)$, for any node i in the graph as,

$$\begin{aligned} \mathcal{N}(i) &= \left\{ j \mid j \in Adj(i) \wedge D_j(r) + w_{ij} \right. \\ &= \left. \min_{k \in Adj(i)} (D_k(r) + w_{ik}) \right\}. \end{aligned}$$

The set $\mathcal{N}(i)$ contains neighboring nodes of i that are on currently estimated shortest paths from node i to r . Now we can define our legitimate state of the system as follows:

Definition 1. The system is in the legitimate state iff

$$\begin{aligned} \forall i \in V : D_i(r) &= S_i(r) \wedge (i = r \wedge P(i) \\ &= NULL \vee i \neq r \wedge P_i \in \mathcal{N}(i)). \end{aligned}$$

Remark 4. The predecessor pointer $P_i \in \mathcal{N}(i)$ means that the predecessor pointer of node i is pointing to one of the nodes in $\mathcal{N}(i)$; exact setting of P_i depends on implementation details.

Remark 5. In any connected symmetric graph, the legitimate state as described above represents a shortest path spanning tree of the graph rooted at the given node r ; also, each node knows its immediate predecessor in the shortest path spanning tree.

Any state which is not legitimate is an illegitimate or unstable state. The purpose of our algorithm is to bring the system back to the stable or legitimate state whenever it enters an illegitimate state due to perturbation. The idea is that whenever the system is in an illegitimate state, at least one node recognizes it and takes corrective action. Each node looks at the states of its adjacent neighbors and checks the local satisfiability of the global legitimate state; if the node is not locally legitimate, it becomes so by adjusting its level variable and the predecessor pointer. More formally, each node i executes the following code:

if $(i = r \wedge (D_i \neq 0 \vee P_i \neq NULL))$ **then**

$D_i = 0 \& P_i = NULL$

else if $\left(i \neq r \wedge (D_i(r) \neq \min_{j \in Adj(i)} (D_j(r) + w_{ij}) \vee P_i \notin \mathcal{N}(i)) \right)$

then $D_i(r) = \min_{j \in Adj(i)} (D_j(r) + w_{ij}) \& P_i = k, k \in \mathcal{N}(i)$

We can now define a *privileged node*: the reference node r is privileged when $D_r(r) \neq 0$ and any other node i is privileged when $D_i(r) \neq \min_{j \in Adj(i)} (D_j(r) + w_{ij})$. It is to be noted that the reference node r may be privileged by perturbation, but once it takes action, it is never privileged again. This is not true for the other nodes.

3.2. Proof of correctness

We show that the SPST protocol does come to a legitimate state in at most n rounds (as defined in Section 2) when the shortest path spanning tree has been constructed in the network graph.

Lemma 1. Starting from a given illegitimate state consider the system state after p rounds; each of the nodes that are yet to be stabilized has $D_i(r) \geq pm_1$.

Proof. This can be easily shown by induction. It is clear that starting from a given illegitimate state the number of stabilized nodes steadily increases until all the nodes in the system are stabilized. Without any loss of generality we can also assume that in the illegitimate

state the D values at nodes are all non-negative. After one round, the node r is stabilized ($D_r(r)$ has been set to zero and it will never be privileged again). For any other node i , $D_i(r)$ will be set to $\min_{j \in \text{Adj}(i)} \{D_j(r) + w_{ij}\} \geq m_1$. Thus, the claim is true for $p = 1$. Assume the claim is true for $p = k$. Consider a node i after the $(k + 1)$ st round. If node i has a minimum path to node r via any of the already stabilized nodes, node i is stabilized and will never be privileged again. If node i has a shortest path to node r via any non-stabilized node, then $D_i(r) \geq D_j(r) + w_{ij} \geq km_1 + m_1 = (k + 1)m_1$, where j is some non-stabilized neighbor of node i after the $(k + 1)$ st round. \square

Lemma 2. Consider a node i which is p hops away from root r (the shortest path from i to r may involve more edges); node i will be stabilized in at most $p \lceil \frac{m_2}{m_1} \rceil$ rounds, after the node r is stabilized.

Proof. When the protocol is invoked in an illegitimate state, any node i has $D_i(r) \geq 0$. The maximum value of the distance of node i from r is pm_2 . Thus, in the worst case, node i will take $p \lceil \frac{m_2}{m_1} \rceil$ rounds to stabilize (Lemma 1). \square

Lemma 3. The upper bound on the number of rounds needed by the entire network to stabilize starting from an arbitrary illegitimate state is given by $\mathcal{D} * \lceil \frac{m_2}{m_1} \rceil + 1$.

Proof. Since any node in the entire system graph is at most \mathcal{D} hops away from the source node r , the maximum number of hops needed by any node is given by $\mathcal{D} * \lceil \frac{m_2}{m_1} \rceil$, after the node r has stabilized. Note that node r takes one round to stabilize and hence the extra round is needed in the worst case. \square

4. Minimal spanning tree (MST) protocol

Let $G = (V, E)$ be an undirected (symmetric) graph (with no self-loops and no parallel edges) representing the distributed system, where V is the set of nodes, $|V| = n$, and E is the set of edges. We use G and V interchangeably to denote the set of nodes of the graph. The minimal spanning tree (MST) of the graph is defined to be a spanning tree of the graph such that the sum of the weights of the edges in the tree is less than or equal to that for all possible spanning trees of the graph. Our objective is to design a self-stabilizing distributed algorithm that constructs the MST of the graph. We begin with an important observation.

Remark 6. If the weights $\{w_{ij}\}$ of a graph are unique (distinct), the graph has a unique MST [HS84].

To design a self-stabilizing algorithm for the MST of a graph, we introduce a new characterization of any path in a given graph.

Definition 2. α -cost of any path from node i to j is defined to be the maximum of the weights of the edges belonging to the path. Ψ_{ij} is defined to be the minimum among the α -cost of all possible paths between the nodes i and j .

Remark 7. We call the path, along which Ψ_{ij} is defined, to be the minimum- α path between nodes i and j ; this should not be confused with the traditional shortest path between nodes i and j . The shortest path is defined to be the path of minimum length where the length of a path is the sum of the weights of the edges on the path. Most significant difference between the two metrics, α -cost and length, of a path, assuming non-zero positive edge weights, is that when a path is augmented by an additional edge, length must increase while α -cost may remain constant.

Remark 8. Consider a graph G with unique edge weights. An edge e_{ij} is in the unique MST if and only if $\Psi_{ij} = w_{ij}$ [MP88].

We use Remarks 6 and 8 to develop our algorithm for MST construction. First, we can safely assume the edge weights to be unique; this is no restriction since if not, we can easily add lexicographic information to make them unique [MP88]. Second, if a distributed algorithm can compute the α_{ij} values for all nodes, we can add an additional data structure Ω_i at each node i that keeps track of the MST edges incident on node i , i.e., $\Omega_i = \{k | \text{the edge } e_{ik} \in \text{MST}\}$. Computing α_{ij} values for all nodes is not similar to the all pairs shortest path problem since the metric α -cost does not have the desirable properties of the metric length (see Remark 7); we cannot use a standard self-stabilizing algorithm for all-pairs shortest path problem. We need to have some additional concepts and data structures to ensure termination of the algorithm in finite time.

For convenience of description and understanding we first develop a self-stabilizing algorithm for minimum α -cost path to a given reference node r in the graph and then generalize the result to solve the MST problem.

4.1. Minimum α -cost path to a given node r

Each node attempts to compute the α -cost of the shortest path (minimum α -cost path) to a given reference node. Call this special node r . Ψ_{ir} denotes the α -cost of the shortest path from node i to node r . Note that for all i , Ψ_{ir} is determined by the topology of the graph and the

weights assigned to the edges. Note that $\Psi_{rr} = 0$ (no self-loops). We use the following notations:

- C : an integer constant such that $C \geq n$.
- $\mathcal{N}(x)$: the set of neighbors of node x .
- $L(i)$: the level of node i , the current estimate of the number of edges on the minimum α -cost path.
- $D(i)$: the current estimate of Ψ_{ir} as known at node i .

Thus each node i maintains two data structures $L(i)$ and $D(i)$ and they determine the *local state* of node i . We assume that $0 \leq L(i) \leq C$; we do not need to consider level values beyond that (even after perturbation), as we can always assume each processor is capable of doing a modulo $(C + 1)$ operation and always keeps the remainder as its level value. The variable $D(i)$ assume an arbitrary value between 0 and some large positive number which we shall call MAX (determined by the length of the registers holding these variables).

Definition 3. For any arbitrary node x , the ordered pair, $S(x) = (D(x), L(x))$ defines the local state of the node x at any given point of time. The vector of all the node states define the global state of the system.

We introduce a total ordering relation between any two arbitrary local states.

Definition 4. Given two local states $S = (D, L)$ and $S' = (D', L')$, S is less than S' or $S < S'$, iff $(D < D') \vee ((D = D') \wedge (L < L'))$, i.e., state tuples are lexicographically ordered.

Definition 5. In any system state, for any arbitrary node x , we define $\mathcal{N}_C(x) = \{y | y \in \mathcal{N}(x), L(y) < C\}$, to be the set of its neighbors with level value $< C$.

Definition 6. In any system state, for any arbitrary node x , $\mathcal{N}_C(x) \neq \emptyset$, we define the following: (1) $\delta_{\min}(x) = \min_{y \in \mathcal{N}_C(x)} \{\max\{w_{xy}, D(y)\}\}$; (2) $\Delta_{\min}(x) = \{y | (y \in \mathcal{N}_C(x)) \wedge (\max\{w_{xy}, D(y)\} = \delta_{\min}(x))\}$; (3) $L_{\min}(x) = \min \{L(y) | y \in \Delta_{\min}(x)\}$.

We make the following immediate observations:

- (a) If the set $\mathcal{N}_C(x)$ for any node x is empty, all neighbors of node x has a level equal to C . The parameters $\delta_{\min}(x)$, $\Delta_{\min}(x)$ and $L_{\min}(x)$ are undefined indicating that the estimates at each neighbor of node x is wrong.
- (b) $\delta_{\min}(x)$ of any node x is a refined estimate of Ψ_{xr} based on the estimates at the neighbors of node x . $\delta_{\min}(x)$ is defined when $\mathcal{N}_C(x) \neq \emptyset$.
- (c) The set $\Delta_{\min}(x)$ denotes the neighbors y of node x such that $\max\{w_{xy}, D(y)\} = \delta_{\min}(x)$. The set $\Delta_{\min}(x)$ is defined and non-empty when $\mathcal{N}_C(x) \neq \emptyset$.
- (d) $L_{\min}(x)$ indicates the minimum of the level values of

the nodes in the set $\Delta_{\min}(x)$. The parameter $L_{\min}(x)$ is defined when $\mathcal{N}_C(x) \neq \emptyset$.

Our objective is to design an algorithm to compute the minimum α -cost of each node to the reference node r , i.e., when the algorithm stabilizes, we will have $D(x) = \Psi_{xr}$ at each node x . Each node x looks at its own state $S(x)$ (the pair $(D(x), L(x))$) and the states of its neighbors and takes action by changing its own level and cost estimate. Our algorithm has a single rule for all the nodes in the graph (actually, the reference node take different action than all other nodes). The rule at node x is as follows:

$$(R) \left\{ \begin{array}{l} \text{if } (x = r) \wedge (L(x) \neq 0 \vee D(x) \neq 0) \\ \quad \text{then } L(x) = 0 \& D(x) = 0; \\ \text{else if } (\mathcal{N}_C(x) = \emptyset) \\ \quad \wedge (D(x) \neq MAX \vee L(x) \neq C) \\ \text{then } D(x) = MAX \& L(x) = C \\ \text{else if } (L(x) \neq L_{\min}(x) + 1) \vee (D(x) \neq \delta_{\min}(x)) \\ \quad \text{then } L(x) = L_{\min}(x) + 1, \& D(x) = \delta_{\min}(x); \end{array} \right.$$

Remark 9. The reference node r is *privileged* if $D(r) \neq 0$ or $L(r) \neq 0$. The reference node may be privileged in an illegitimate state, but once it takes an action, it becomes unprivileged and can never be privileged again.

Remark 10. Any other node x , with $\mathcal{N}_C(x) = \emptyset$ is *privileged* if $(D(x) \neq MAX \vee L(x) \neq C)$; any node x , with $\mathcal{N}_C(x) \neq \emptyset$ is *privileged* if $L(x) \neq L_{\min}(x) + 1 \vee D(x) \neq \delta_{\min}(x)$. Note that any node x , $x \neq r$, is privileged and takes action, it becomes unprivileged, but can be privileged again later (only after at least one move by one of its neighbors).

Remark 11. Given any arbitrary initial system state, the number of all possible distinct local states that any node can have subsequently is finite (L values can range over $0 \dots C - 1$ and the D values can range over the edge weights and the initial D values at the nodes). Thus, the number of all possible global system states is also finite.

Definition 7. Any global system state, when no node is privileged, is called a legitimate state; any other state is illegitimate.

Remark 12. In a legitimate state, $L(r) = D(r) = 0$.

Lemma 4. In a legitimate state, any node x , $x \neq r$, with $L(x) < C$ has $\mathcal{N}_C(x) \neq \emptyset$ and has at least one neighbor y such that $L(y) = L(x) - 1$.

Proof. For any unprivileged node x , with $L(x) < C$, we have $L(x) = L_{\min}(x) + 1$ and since $L(x) < C$, we get

$L_{\min}(x) < C \Rightarrow \mathcal{N}_C(x) \neq \emptyset$. We also have that $L_{\min}(x) = L(x) - 1$ and since $L(x) < C$, there exists at least one neighbor y of node x such that $L(y) = L(x) - 1$. \square

Lemma 5. *In a legitimate state, when no node is privileged, for any arbitrary node x , $L(x) < C$.*

Proof. In a legitimate state, the reference node r has $L(r) = 0$. Assume that a node x has $L(x) = C$; since x is unprivileged, $\mathcal{N}_C(x) = \emptyset$. Consider the subset of nodes in graph G with level C . This subset forms a subgraph G' of G . Since G is connected and $r \notin G'$, there must be at least one node $y \in G'$ such that $\mathcal{N}_C(y) \neq \emptyset$ and since this y is unprivileged, there exists a node z such that $L(z) = L(y) - 1 = C - 1$. Then, by repeated application of the Lemma 4, there must be at least one node each with level values $C - 1, C - 2, \dots, 0$. This is a contradiction since $C \geq n$ where n is the number of nodes in the graph. \square

Corollary 1. *For some integer m , $m < C$, (m denotes the highest level of a node in a legitimate state), the set of nodes in the graph is given by $\bigcup_{0 \leq k \leq m} R(k)$, where $R(k)$ is the set of nodes with level k .*

Lemma 6. *In a legitimate state, (1) $R(0) = \{r\}$; (2) for each node $x \in R(k)$, $1 \leq k \leq m$, there exists a node $y \in R(k - 1)$ such that $D(x) = \max \{D(y), w_{xy}\}$.*

Proof. (1) Clearly, $R(0)$ contains the reference node r since in a legitimate state $L(r) = 0$. Assume $R(0)$ contains another node x . Since x is not privileged and is not the reference node, $L(x) = L_{\min}(x) + 1$ and since levels cannot be negative, $L(x) > 0$; thus, $R(0)$ cannot contain x .

(2) Since node x is not privileged, $L_{\min}(x) = L(x) - 1$ and $D(x) = \delta_{\min}(x)$, i.e., there exists a node y , such that $L(y) = L(x) - 1$ (thus, $y \in R(k - 1)$) and $\delta_{\min}(x) = \max \{D(y), w_{xy}\}$. \square

Theorem 1. *In a legitimate state, when no node is privileged, for any arbitrary node x , we have $D(x) = \Psi_{xr}$.*

Proof. Consider any path $r = y_0, y_1, \dots, y_\ell = x$ from the reference node r to any arbitrary node x . The α -cost of this path is given by $w = \max \{w(y_i, y_{i+1}) | i = 0, \dots, \ell - 1\}$. Also, since no node is privileged, $D(y_0) = 0$, and for all i , $i = 1, \dots, \ell$, $D(y_i) = \delta_{\min}(y_i) \leq \max \{D(y_{i-1}), w(y_{i-1}, y_i)\} \leq w$. Thus, we have proved that for any arbitrary node x , $D(x) \leq \Psi_{xr}$.

To prove $D(x) \geq \Psi_{xr}$, we use induction. Clearly the claim holds for the node r in $R(0)$. Assume the claim hold for nodes in $R(k)$. Consider any arbitrary node x in $R(k + 1)$. By Lemma 6, there exists a node y in $R(k)$ such that $D(x) = \max \{D(y), w_{xy}\}$. Since $D(y) = \Psi_{yr}$ (i.e., there exists a path from node y to node r with

α -cost $D(y)$), there is a path from node x to r with cost $D(x)$, i.e., $D(x) \geq \Psi_{xr}$. \square

4.2. Proof of correctness

Definition 8. A node is called *stabilized* if the node cannot make a further move until a legitimate system state is reached. The set of stabilized nodes after round p is denoted by $V_{\mathcal{G}}^p$.

Lemma 7. *After the first round,*

- (1) *The reference node r is stabilized, i.e., $L(r) = 0$ and $D(r) = 0$ and it will not move again.*
- (2) *Any other node $i (\neq r)$ has $L(i)$ set to C or less and $D(i)$ set to MAX or less.*
- (3) $|V_{\mathcal{G}}^1| \geq 1$.

Proof. (1) follows from the first clause of the algorithm (and Remark 9); (2) follows from the second clause and (3) follows from (1). \square

Lemma 8. *At the end of round p , we have $|V_{\mathcal{G}}^p| \geq p$.*

Proof. We prove this by induction. After the second round, consider the node i whose Ψ_{ir} path does not contain any other node (at least one such node must exist since the root node r is connected to the rest of the tree); node i is stabilized at the end of the second round. Consider $V_{\mathcal{G}}^k$ after round k . Consider the node i from the set of nodes $V - V_{\mathcal{G}}^k$ whose Ψ_{ir} path does not contain any node from the set $V - V_{\mathcal{G}}^k$ (at least one such node must exist since the tree in $V - V_{\mathcal{G}}^k$ to the rest of the spanning tree; node i will be stabilized after round $k + 1$). The claim follows. \square

Theorem 2. *The protocol terminates, i.e., the system reaches a legitimate state after at most n rounds, where n is the number of nodes in the graph.*

Proof. The number of stabilized nodes (the nodes that will not move again) at the end of n th round is at least n (Lemma 8). Since the number of nodes in the graph is n , the claim follows. \square

4.3. MST protocol

We can now generalize the algorithm in the previous section to compute the minimum α -cost paths to all nodes and thereby compute the MST of the graph. Instead of the simple local variable $D(i)$, each node i

now maintains a local array $D_i[1..n]$ and instead of the simple local variable $L(i)$, each node i now maintains a local array $L_i[1..n]$. The value of $D_i[j]$, for all $i, j \in V$, at any system state gives the cost of the minimum α -cost path from node i to j in that system state. Similarly, the value of $L_i[j]$ is the value of the level of node i with respect to the implicit tree rooted at node j . The contents of the arrays $D_i[]$ and $L_i[]$ denote the local state of the node i and the union of all local states defines the global system state. Ψ_{ij} denotes the cost of the minimum α -cost path from node i to node j for all i and j . Note that $\Psi_{ii} = 0$ for all i . Each node behaves as a special (reference) node when it attempts to compute the α -cost to itself; it unconditionally sets that value to 0. The data structure Ω_i at each node i keeps track of the MST edges incident on node i .

We also need to root the MST at the given source node r , i.e., each node i will know its predecessor in the MST (using a variable P_i). To do this, we need to have a variable $Level_i$ at each node i to indicate its level in the MST. Obviously for the source node r , we should have $P_r = A$ and $Level_r = 0$. We now present the self-stabilizing algorithm to compute the MST. Every node in the system has the same uniform rule. The rule at node i is as follows:

$$\begin{array}{l}
 \text{MST} \left\{ \begin{array}{l}
 \forall j = 1, \dots, n \text{ do} \\
 \text{if } ((j = i) \wedge (D_i(j) \neq 0) \vee (L_i(j) \neq 0)) \\
 \quad \text{then } L_i(j) = 0 \& D_i(j) = 0; \\
 \text{else if } ((j \neq i) \wedge (\mathcal{N}_C(i) = \emptyset) \\
 \quad \wedge (D_i(j) \neq MAX \vee L_i(j) \neq C) \\
 \quad \text{then } D_i(j) = MAX \& L_i(j) = C \\
 \text{else if } ((j \neq i) \wedge (L_i(j) \neq L_{\min}(j) + 1) \\
 \quad \vee (D_i(j) \neq \delta_{\min}(j))) \\
 \quad \text{then } L_i(j) = L_{\min}(j) + 1 \& D_i(j) = \delta_{\min}(j) \& \\
 \quad \quad \Omega_i = \{k | k \in \mathcal{N}(i) \wedge w_{ik} = D_i(k)\};
 \end{array} \right. \\
 \\
 \text{Root at } \left\{ \begin{array}{l}
 \text{if } (i = r) \wedge (Level_i \neq 0 \vee P_i \neq A) \\
 \quad \text{then } Level_i = 0 \& P_i = A) \\
 \text{else } Level_i = Level_y | (y \in \Omega_i \\
 \quad \wedge Level_y \leq Level_k, \quad k \in \Omega_i) \& P_i = y
 \end{array} \right.
 \end{array}$$

Lemma 9. *Construction of the MST is completed by at most n rounds.*

Proof. The proof directly follows from Theorem 2 and the fact that in the MST protocol each node i concurrently attempts to construct the Ψ_{ij} path for all k , $1 \leq k \leq n$; construction of each such path takes at most n rounds and the MST is constructed when all such paths are stabilized. \square

Lemma 10. *The process of rooting the MST at the source node r takes at most n rounds, after the MST has stabilized.*

Proof. The root is established in the first round and in each successive round the nodes at the successive levels of the MST will have their predecessor pointer and the *Level* value stabilized. The rooting of the MST at any node r will take at most h rounds (after the MST has been stabilized), where h is the height of the MST and in the worst case $h = n$. \square

Theorem 3. *The MST protocol stabilizes in at most $2n$ rounds.*

Proof. The proof follows directly from Lemmas 9 and 10. \square

5. Multi-cast protocols

Our protocol for fault tolerant maintenance of the multi-cast tree for a given source node (we call it root node r) and its multi-cast group consists of 2 logical steps: (1) construction of the shortest path spanning tree (or the minimal spanning tree) of the mobile network graph in presence of the topology change due to node mobility (establishing unique parent pointer for each node in the SPST or MST); (2) pruning from the SPST the nodes that are not needed to send the message to the multi-cast group members. The protocols described in the previous sections maintain the SPST or the MST in a fault tolerant way (that accommodates the topology change due to node mobility) as well as maintain the knowledge of the tree in a distributed way; each node knows its unique parent pointer). In this section we describe the protocol to prune the SPST/MST and build the multi-cast tree.

The multi-cast source node r needs to send the message to the members of the arbitrary multi-cast group. Each node in the network knows whether it is a member of the multi-cast group (IS_Member_i is true). Note that even if a node is not a member of the multi-cast group, it will need to transmit the message to its successors iff any of its successors belong to the multi-cast group. In the rooted SPST/MST, each node i can determine if it is a leaf node in the SPST (it has no neighbor node j such that $P_j = i$); in this case, node i will set $Flag_i$ variable to 1 if IS_Member_i is true and to 0 otherwise. Any other node i (i is not a leaf node in the SPST/MST) will look at all its successors in the SPST/MST and will set its $Flag_i$ to 1 iff at least one of its successors either has a $Flag$ of 1 or is a member of the multi-cast group or node i is a member of the multi-cast group. After this process stabilizes, each node i , when it

receives the multi-cast message from its parent in the tree, knows that it needs to forward the message to its successors if $Flag_i$ is 1. Note that the nodes with $Flag_i$ value 1 constitute the multi-cast tree (although not all the nodes in the multi-cast tree are necessarily members of the multi-cast group). Now we can state the complete protocol to maintain the multi-cast tree:

SPST

or {

MST

Multi-cast Tree

$$\left\{ \begin{array}{l} \text{if } Flag_i \neq \vee_{k \in Adj(i)} ((P_k = i) \wedge (IS_Member_k \vee Flag_k)) \\ \quad \text{then } Flag_i = \vee_{k \in Adj(i)} ((P_k = i) \wedge \\ \quad \quad (IS_Member_k \vee Flag_k)) \end{array} \right.$$

In the previous sections we have shown the correctness of the protocols for maintaining the SPST or MST and provided its complexity analysis. In this section, we show the correctness of the protocol to generate the multi-cast tree assuming that the SPST or MST has been stabilized and analyze its complexity.

Lemma 11. *Starting from any illegitimate state the protocol correctly sets the Flag for each node which is a member of the multi-cast group, in at most $n - 1$ rounds after the SPST or MST protocol has stabilized.*

Proof. Consider the first round after the SPST or MST protocol has stabilized. Any leaf node i of the shortest path spanning tree, who is a member of the multi-cast group (IS_Member flag is 1) will set its flag variable $Flag_i$ to 1; once they are set, they will never be reset. In the next round, any node j one hop away from any leaf node, will look at its successors (in the SPST or MST)

and will set its $Flag_j$ appropriately and will never be changed in any subsequent round. Since a node can be at most $n - 1$ hops away from any leaf node in a spanning tree of a graph of n nodes, the claim easily follows. \square

Theorem 4. *Starting from any illegitimate state, the entire protocol stabilizes to a valid multi-cast tree in at most $\mathcal{D} * \lceil \frac{m_2}{m_1} \rceil + n$ rounds.*

Proof. The proof readily follows from Lemmas 3 and 11. \square

6. Preliminary simulation results

We conducted preliminary simulation experiments to investigate the convergence time of the MST and the SPST protocol. Network Simulator (NS-ver 2.1b8a) was used to simulate the protocols. Initially the nodes are placed in arbitrary points inside a grid. Each node is assigned a random velocity with which it moves inside the grid. Whether a node is a multicast node or not was also set randomly. The movement pattern of node follows the NS node movement pattern without random motion viz. a node tries to reach the destination through a shortest path with the given velocity. The beacon messages are sent periodically. The simulation was run for 5, 10, 20, 30 and 50 nodes for both protocols. The grid size used was 800×800 for 5, 10, 20, 30 nodes and 1000×1000 for 50 nodes, beacon interval was chosen to be 6 s for all nodes. Experiments were repeated large number of times for each class of nodes. The results are shown in Fig. 1.

Number Of nodes	Protocol	Number of rounds to stabilize without any node movement	Number of rounds to stabilize with node movement	
			Min	Max
5	SPST	3	1	2
5	MST	3	2	3
10	SPST	3	1	7
10	MST	3	2	6
20	SPST	3	1	8
20	MST	3	2	7
30	SPST	4	2	11
30	MST	4	3	15
50	SPST	4	3	22
50	MST	6	3	19

Fig. 1. Preliminary simulation results.

7. Conclusion

We have proposed two multi-cast protocols for ad hoc networks; the protocols are mobility tolerant in the sense that they can adapt to the dynamic change in topology in ad hoc networks due to user mobility. The protocols are based on existing self-stabilizing distributed algorithms for SPST and MST computation in networks. We have provided detailed complexity analysis of the protocols in the context of ad hoc networks and have shown that both protocols converge in time linear in the number of participating nodes in the network. Our results show that the concept of self-stabilization can be efficiently used in designing fault tolerant protocols for ad hoc networks and we expect this will lead to design of fault tolerant protocols for other primitives in ad hoc networks. We have provided very preliminary simulation results regarding performance of the protocols; more extensive experiments are underway to evaluate the typical average behavior of the protocols. It would also be interesting to relax some of the assumptions in our system model to investigate wider applicability of the concept of self-stabilization.

Acknowledgments

This research was supported by NSF Award # ANI-0073409. The authors wish to thank the anonymous reviewers for many helpful comments that improved the presentation of the paper.

References

- [BFC93] T. Ballardie, P. Francis, J. Crowcroft, Core based trees (CBT): an architecture for scalable inter domain multi-cast routing, in: SIGCOMM, ACM, 1993, pp. 85–95.
- [CS94] S. Chandrasekar, P.K. Srimani, A self-stabilizing distributed algorithm for all-pairs shortest path problem, *Parallel Algorithms Appl.* 4 (1&2) (1994) 125–137.
- [DEF96] S.E. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, L. Wei, The PIM architecture for wide-area multicast routing, *IEEE/ACM Trans. Networking* 4 (2) (1996) 153–162.
- [GBS00] S.K.S. Gupta, A. Bouabdallah, P.K. Srimani, Self-stabilizing protocol for shortest path tree for multicast routing in mobile networks, *Lecture Notes in Computer Science*, vol. 1900, Springer, Berlin, April 2000, pp. 600–604.
- [GS99] S.K.S. Gupta, P.K. Srimani, Using self-stabilization to design adaptive multicast protocol for mobile ad hoc networks, in: *Proceedings of the DIMACS Workshop on Mobile Networks and Computing*, Rutgers University, NJ, March 1999.
- [HDL02] Z. Haas, J. Deng, B. Liang, P. Papadimitratos, S. Sajama, *Wireless ad hoc networks* in: J. Proakis (Ed.), *Encyclopedia of Telecommunications*, Wiley, New York, 2002.
- [HS84] E. Horowitz, S. Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press, Rockville, MD, 1984.
- [Kar72] R.M. Karp, Reducibility among combinatorial problems, in: *Complexity of Computer Computations*, Plenum Press, New York, 1972.
- [Kes97] S. Keshav, *An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network*, Addison-Wesley, Reading, MA, 1997.
- [KZ89] A. Khanna, J. Zinky, The revised arpanet routing metric, in: *ACM SIGCOMM'89*, August 1989.
- [MP88] B.M. Maggs, S.A. Plotkin, Minimum-cost spanning tree as a path finding problem, *Inform. Process. Lett.* 26 (1988) 291–293.
- [PD96] L.L. Peterson, B.S. Davie, *Computer Networks*, 2nd Edition, Morgan Kaufmann, Los Altos, CA, 1996.
- [Per01] C. Perking (Ed.), *Ad Hoc Networking*, Addison-Wesley, Reading, MA, 2001.
- [Sch93] M. Schneider, Self-stabilization, *ACM Comput. Surveys* 25 (1) (1993) 45–67.
- [Sto02] I. Stojmenovic (Ed.) *Handbook of Wireless Networks and Mobile Computing*, Wiley, New York, 2002.