To appear in the Proceedings of the Fifth IPDPS Workshop on Advances in Parallel and Distributed Computational Models, Nice, France, April 22-26, 2003; http://www.ipdps.org/ipdps2003/

Self-Stabilizing Protocols for Maximal Matching and Maximal Independent Sets for Ad Hoc Networks*

Wayne Goddard, Stephen T. Hedetniemi David P. Jacobs and Pradip K Srimani
Department of Computer Science
Clemson University
Clemson, SC 29634-0974

Abstract

We propose two distributed algorithms to maintain, respectively, a maximal matching and a maximal independent set in a given ad hoc network; 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 global predicates. 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 their 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 an adversary deamon used in proving the convergence and correctness of self-stabilizing distributed algorithms in general.

Keywords: Self-stabilizing protocol, distributed system, multi-cast protocol, fault tolerance, convergence, system graph.

1 Introduction

Most essential services for networked distributed systems (mobile or wired) involve maintaining a global predicate over the entire network (defined by some invariance relation on the global state of the network) by using local knowledge at each participating node. For example, a minimal spanning tree must be maintained to minimize latency and bandwidth requirements of multicast/broadcast messages or to implement echo-based distributed algorithms [1, 2, 3, 4]; a minimal dominating set must be maintained to optimize the number and the locations of the resource centers in a network [5]; an (r, d) configuration must be maintained in a network where various resources must be allocated but all nodes have a fixed capacity r [6]; a minimal coloring of the nodes must be maintained [7].

In this paper we propose two distributed algorithms to maintain respectively a maximal matching and a maximal independent set in a given ad hoc network. 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 uses *self-stabilization* [8, 9, 10] to design the fault-tolerant distributed algorithms.

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 beacon message provides an inexpensive way of periodically exchanging additional information between neighboring nodes. In our algorithm, a node takes action after receiving beacon messages (along with algorithm related information) from all the neighboring nodes. 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. The beacon messages provide information about its neighbor nodes synchronously (at specific time intervals). Thus, the paradigm to analyze the complexity of the self-stabilizing algorithms in 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.

*This work has been supported by NSF grant # ANI-0073409 and NSF grant # ANI-0218495

Similar paradigms have been used in [11, 12, 13, 14].

2 System Model

We make the following assumptions about the system. A 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. The logical links between two neighboring nodes are assumed to be bounded and FIFO. The link layer protocol informs the upper layer of any creation/deletion of logical links using the *neighbor discovery protocol* described below.

Each node periodically (at intervals of t_b) broadcasts a *beacon* message. This forms the basis of the *neighbor discovery protocol*. When node i receives the beacon signal from node j which is not in its neighbors list $neighbors(i)$, it adds j to its neighbors list, 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 is used by node i to compute the cost of the link (i, j) as well as 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 time-outs, 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. We also assume each node is assigned a unique ID.

3 Maximal Matching

Given an undirected graph $G = (V, E)$, a *matching* is defined to be a subset $M \subseteq E$ of pairwise disjoint edges. That is, no two edges in M are incident with the same node. A *matching* M is maximal if there does not exist another matching M' such that $M' \supset M$.

We present a synchronous model self-stabilizing protocol for finding a maximal matching in an arbitrary network. While the central daemon algorithm of [15] may be converetd into a synchronous model protocol using the techniques of [11, 16], the resulting protocol is not as fast. The pseudocode of our proposed algorithm is shown in Figure 1. Each node i maintains a single pointer variable which is either null, denoted $i \rightarrow \Lambda$, or points to one of its neighbors j, denoted $i \rightarrow j$.

We say a node i is *matched* if there exists another node $j \in N(i)$ such that $i \rightarrow j \wedge j \rightarrow i$ (denoted by $i \leftrightarrow j$). In rule **R1**, a node i, whose pointer is null, may select a node $j \in N(i)$, among those that are pointing to it, and become matched with j. Informally, we say that i *accepts* a proposal made by j. Rule **R2** allows a node i, (whose pointer is null and has no neighbors currently pointing to it), to point to a neighbor j having a null pointer. Note that i may not select an arbitrary neighbor, but rather the neighbor with null pointer and with minimum ID. Informally, we say that i *proposes* to j. Rule **R3** is executed by i when i is pointing to a neighbor j which in turn is pointing to another node $k \neq i$. In this case, i sets its pointer to null, and we say that it *backs-off*.

The *global system state* S_t at time t is defined to be the union of the local states (values of the pointer variables) of each node i at time t. Thus for any time instant t,

$$S^t \xrightarrow{\quad m_t \quad} S^{t+1}$$

where S^0 denotes the arbitrary initial (starting) state. We define the following type classification of nodes of a network in any global system state S^t:["M" is for "matched", "P" is for "pointing" and "A" is for "aloof"]

$$\begin{aligned}
M^t &\stackrel{\text{def}}{=} \{i \in V : i \leftrightarrow j \text{ for some } j\} \\
A^t &\stackrel{\text{def}}{=} \{i : i \rightarrow \Lambda\} \\
A_0^t &\stackrel{\text{def}}{=} \{i : i \in A^t \wedge \nexists j \in N(i) : j \rightarrow i\} \\
A_1^t &\stackrel{\text{def}}{=} \{i : i \in A^t \wedge \exists j \in N(i) : j \rightarrow i\} \\
P^t &\stackrel{\text{def}}{=} \{i : i \rightarrow j : j \neq \Lambda \wedge j \not\rightarrow i\} \\
PA_1^t &\stackrel{\text{def}}{=} \{i \in P^t : i \rightarrow j : j \in A_1^t\} \\
PM^t &\stackrel{\text{def}}{=} \{i \in P^t : i \rightarrow j : j \in M^t\} \\
PP^t &\stackrel{\text{def}}{=} \{i \in P^t : i \rightarrow j : j \in P^t\}
\end{aligned}$$

Note that each time t, $\{M^t, A^t, P^t\}$ defines a (weak) partition of V, $\{A_0^t, A_1^t\}$ defines a (weak) partition of A^t,

R1: if $(i \to \Lambda) \wedge (\exists j \in N(i) : j \to i)$
then $i \to j$ [accept proposal]
R2: if $(i \to \Lambda) \wedge (\forall k \in N(i) : k \not\to i) \wedge (\exists j \in N(i) : j \to \Lambda)$
then $i \to \min\{j \in N(i) : j \to \Lambda\}$ [make proposal]
R3: if $(i \to j \wedge j \to k \neq \Lambda \wedge k \neq i)$
then $i \to \Lambda$ [back-off]

Figure 1. Algorithm SMM: Synchronous Maximal Matching

and $\{PA_1^t, PM^t, PP^t\}$ defines a (weak) partition of P^t. Figure 2 illustrates these sets. The situation is simpler than it appears, for as we will see in Lemma 7, sets A_1^t and PA_1^t are empty, except possibly at initialization ($t = 0$).

Lemma 1 *For any time t,* $M^t \subseteq M^{t+1}$.

Proof : If a node $i \in M^t$, then $i \leftrightarrow j$ for some j. Clearly, neither i nor j is privileged at time t, so at time $t+1$ they remain matched. □

Lemma 2 *For any time t,* $PM^t \subseteq A_0^{t+1}$.

Proof : If node $i \in PM^t$, then node i must execute rule **R3** since i is pointing to a node $j \in M^t$, and $j \to k \neq i$. Thus, at time $t+1$, i will have a null pointer. Also, no neighbor of node i can propose to i at time t, since i does not have a null pointer. □

Lemma 3 *For any time t,* $PP^t \subseteq A_0^{t+1}$.

Proof : The proof is identical to that of Lemma 2. □

Lemma 4 *For any time t,* $PA_1^t \subseteq M^{t+1} \cup PM^{t+1}$.

Proof : Let $i \in PA_1^t$. Then at time t, $i \to j$ for some $j \in A_1^t$. At time t, node i cannot make a move, and node j must execute **R1**. If j accepts the proposal of i, then at time $t+1$, $i \in M^{t+1}$. On the other hand, if j accepts the proposal of another node, then at time $t+1$, $i \in PM^{t+1}$. □

Lemma 5 *If* $i \in A_1^t$ *then* $i \in M^{t+1}$, *and* $|M^{t+1}| \geq |M^t| + 2$.

Proof : If node $i \in A_1^t$, then at time t, node i will execute rule **R1**, setting its pointer to some node $j \in PA_1^t$; node j cannot make a move at time t and both node i and j enter into the subset M^{t+1}. □

Lemma 6 *For any time t,* $A_0^t \subseteq A_0^{t+1} \cup PM^{t+1} \cup M^{t+1} \cup PP^{t+1}$.

Proof : If $i \in A_0^t$, then i will move at time t if and only if there is a neighbor j having a null pointer. If, at time t, i moves and points to j, j will also point to some node. Whether j matches with i, or matches with some other node, or does not match determines whether $i \in M^{t+1}$, or $i \in PM^{t+1}$, or $i \in PP^{t+1}$. □

We have summarized Lemmas 2–6 in Figure 3. The arrows may be interpreted as state changes that a node *must* make in one time unit. Multiple outgoing arrows suggest alternate possible membership changes. Since there are no incoming arrows into A_1 or PA_1 we can conclude that

Lemma 7 *After time* $t = 0$, *the sets* A_1^t *and* PA_1^t *are always empty.*

Lemma 8 *If for any time t, the system is stable (i.e. no node can move), then the subset of nodes* M^t *defines a maximal matching, and every node not in* M^t *is in* A_0^t.

Proof : It is clear that M^t represents a matching since each node has only one pointer, and the nodes in M^t are pointing to each other in pairs (thus denoting the edges in the matching). Since the system is stable, it follows, by Lemma 2, Lemma 3, Lemma 4, and Lemma 5 that all unmatched nodes are in A_0^t. But M^t must be a maximal matching. For if there are two adjacent nodes in A_0^t it follows that rule **R2** is applicable to both nodes, contradicting the stability of the system. □

Lemma 9 *If any node in* A_0^t *makes a move at time t, then* $|M^{t+1}| \geq |M^t| + 2$.

Proof : The only move possible by a node in A_0^t is to execute rule **R2**. Among all nodes $i \in A_0^t$ moving at time t, assume that i has smallest ID. If i makes a proposal to some node j, then node j must be in A_1^t or A_0^t. If $j \in A_1^t$, then by Lemma 5, the cardinality of M increases by at least two. On the other hand, if i proposes to $j \in A_0^t$, then j must also propose to i because of the minimality of the i, thus increasing the cardinality of M by two. □

Lemma 10 *Let* $t > 0$. *If a move is made at time* $t+1$, *then* $|M^{t+2}| \geq |M^t| + 2$.

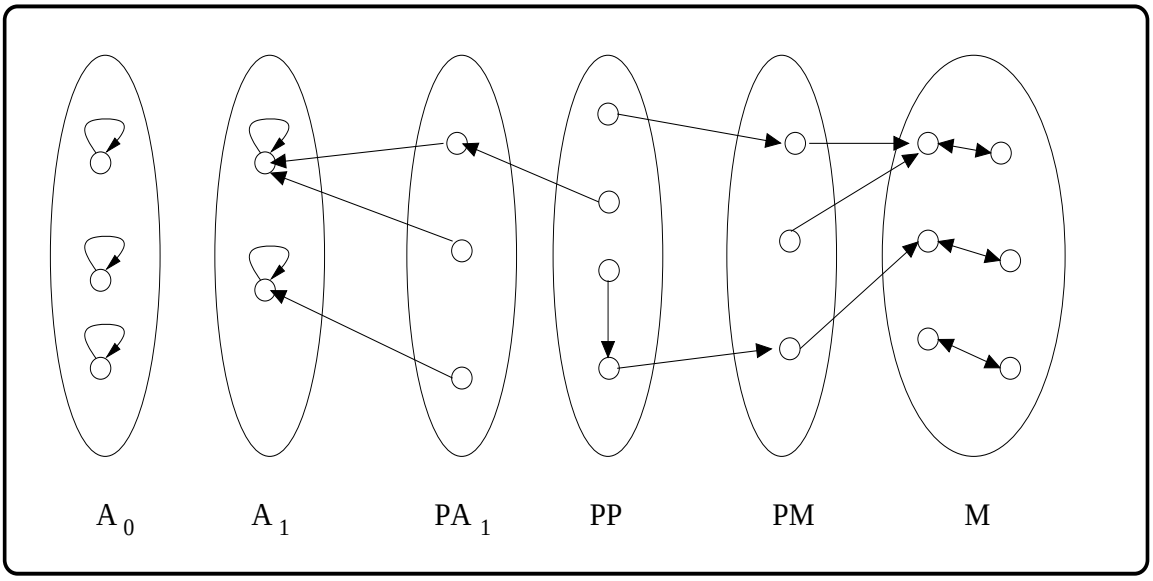

Figure 2. Possible Types of Nodes in any Global State S^t

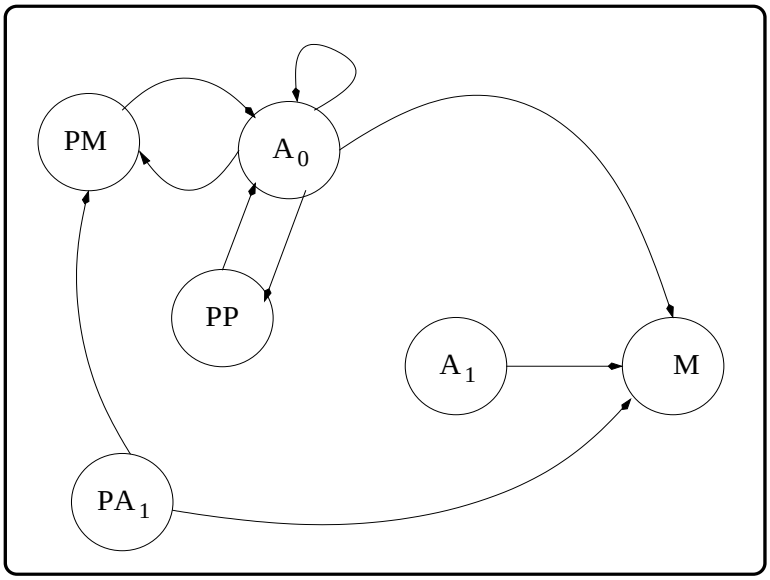

Figure 3. Type Transition Diagram of Nodes

Proof : By Lemma 7, at time t, all nodes that move are in PM, PP, or A_0. By our assumption there must be nodes that move at times t and $t+1$. If some member of A_0^t moves at time t, then by Lemma 9 the result follows. If no member of A_0^t moves at time t, then at time $t+1$ $PM \cup PP = \emptyset$. Hence there must be a node A_0^{t+1} that moves at time $t+1$. Again, by Lemma 9 the result follows. □

Theorem 1 *Starting from an arbitrary state, the algorithm SMM stabilizes and produces a maximal matching in at most $n+1$ rounds, where n is the number of nodes in the network.*

Proof : Lemma 10 shows that after $2k+1$ rounds, at time $t = 2k+1$, there are at least $2k$ nodes in M. Since $2k \leq n$, it follows that $2k+1 \leq n+1$. □

It is interesting to note that in rule **R2** of Algorithm SMM, it is necessary that i select a minimum neighbor j, rather than an arbitrary neighbor. For if we were to omit this requirement, the algorithm may not stabilize: Consider a four cycle, with all pointers initially null, which repeatedly select their clockwise neighbor using rule **R2**, and then execute rule **R3**.

4 Maximal Independent Set

A set S of nodes is *independent* if no two members in S are adjacent. In this section we present a synchronous model self-stabilizing protocol for finding a maximal independent set. We assume that no two neighbors have the same ID. Algorithm SIS, shown in Figure 4, has only two simple rules, each of which assumes that every node can compare its ID with that of its neighbor. Thus, if $i > j$, then we say that j is bigger than i. The following lemmas are straightforward, and we omit the proofs.

Lemma 11 *If at any time t, the set of nodes with $x(i) = 1$ does not form an independent set then at least one node will make an* **R2** *move at any time $t+1$.*

Lemma 12 *If at any time, the set of nodes with $x(i) = 1$ forms an independent set which is not maximal, then at least one node will make an* **R1** *move.*

Lemma 13 *If Algorithm SMI stabilizes, then $\{i | x(i) = 1\}$ forms a maximal independent set.*

Theorem 2 *Algorithm 4 stabilizes in $O(n)$ round.*

Proof : (Sketch) At time $t = 1$, all largest nodes i will set $x(i) = 1$. At time $t = 2$, all neighboring nodes j will be permanently set to $x(j) = 0$. Similarly, all largest nodes $k \in V - N[i]$ will be set $x(k) = 1$ by time $t = 3$, and by time $t = 4$ every node in $(V - N[i]) \cap N(k)$ will be permanently set to 0. This process will continue until all nodes are stable, in at most n rounds. □

5 Conclusions

It can be shown that problems that are solvable with self-stabilizing algorithms using the centralized model, are generally solvable using the synchronous model. However, there is no guarantee that the synchronous algorithm will be fast. We have shown that for both maximal matching and maximal independent set, a fast synchronous algorithm is possible.

References

[1] S Dolev, DK Pradhan, and JL Welch. Modified tree structure for location management in mobile environments. *Computer Communications*, 19:335–345, 1996.

[2] S Dolev and JL Welch. Crash resilient communication in dynamic networks. *IEEE Transactions on Computers*, 46:14–26, 1997.

[3] H Abu-Amara, B Coan, S Dolev, A Kanevsky, and JL Welch. Self-stabilizing topology maintenance protocols for high-speed networks. *IEEE/ACM Transactions on Networking*, 4(6):902–912, 1996.

[4] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. McGraw Hill, 1998.

[5] T.W. Haynes, S.T. Hedetniemi, and P.J. Slater. *Fundamentals of Domination in Graphs*. Marcel Dekker, 1998.

[6] S. Fujita, T. Kameda, and M. Yamashita. A resource assignment problem on graphs. In *Proceedings of the 6th International Symposium on Algorithms and Computation*, pages 418–427, Cairns, Australia, December 1995.

[7] S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani. Fault tolerant distributed coloring algorithms that stabilize in linear time. In *Proceedings of the IPDPS-2002 Workshop on Advances in Parallel and Distributed Computational Models*, pages 1–5, 2002.

[8] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, November 1974.

[9] E. W. Dijkstra. A belated proof of self-stabilization. *Distributed Computing*, 1(1):5–6, 1986.

[10] S Dolev. *Self-Stabilization*. MIT Press, 2000.

R1: if $(x(i) = 0) \wedge (\nexists j \in N(i) : j > i \wedge x(j) = 1)$
then $x(i) = 1$ [enter the set]

R2: if $(x(i) = 1) \wedge (\exists j \in N(i) : j > i \wedge x(j) = 1)$
then $x(i) = 0$ [leave the set]

Figure 4. Algorithm SMI: Synchronous Maximal Independent set

[11] Y Afek and S Dolev. Local stabilizer. In *Proceedings of the 5th Israeli Symposium on Theory of Computing and Systems*, pages 74–84, 1997.

[12] S. Shukla, D. Rosenkrantz, and S. Ravi. Developing self-stabilizing coloring algorithms via systematic randomization. *Proc. Internat. Workshop on Parallel Processing*, pages 668–673, 1994.

[13] SKS Gupta and PK 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*, pages 67–84, Rutgers University, NJ, 1999.

[14] SKS Gupta, A Bouabdallah, and PK Srimani. Self-stabilizing protocol for shortest path tree for multi-cast routing in mobile networks (research note). In *Euro-Par'00 Parallel Processing, Proceedings LNCS:1900*, pages 600–604, 2000.

[15] Su-Chu Hsu and Singh-Tsaan Huang. A self-stabilizing algorithm for maximal matching. *Inform. Process. Lett.*, 43:77–81, 1992.

[16] J Beauquier, AK Datta, M Gradinariu, and F Magniette. Self-stabilizing local mutual exclusion and daemon refinement. In *DISC00 Distributed Computing 14th International Symposium, Springer LNCS:1914*, pages 223–237, 2000.