Locally-Constructed Trees for Adhoc Routing

Ricardo Marcelín-Jiménez [*]

calu@xanum.uam.mx
Electrical Eng. Dept., UAMI;
Atlixco 186; 09340 México D.F., MEXICO

Abstract. We present a family of self-stabilizing distributed algorithms to built a spanning tree on the underlaying communications graph of an adhoc wireless network. Next, based on this principle, we show how to construct two overlaying trees which are suitable for routing tasks.

1 Introduction

Routing using local information has been considered an alternative approach to cope with the main drawbacks of traditional strategies, such as scalability or fault-tolerance [1], [2], [4], [5], [8], [11]. Most of the routing algorithms using local position information, perform a preliminar process on the underlying graph, in order to produce a planar one where working requirements are met [7], [9].

We present a routing algorithm for adhoc wireless nets where each node knows its coordinates on the plane, and the relative coordinates of its neighbors in order to built up two overlaying spanning trees where routing operations have place. Each tree is a self-reconfigurable structure which is able to tolerate links and node failures.

The rest of this work includes the following parts: section 2 presents the theoretical framework that supports our proposal, section 3 introduces our general distributed spanning tree algorithm and shows how routing tasks can be implemented on top of two overlaying structures of this kind, section 4 offers some performance metrics about 4 different types of construction algorithms, based on the principles developed in section 3, finally section 5 is a discussion about the applications and directions for further work.

2 Assumptions and Models

This work deals with distributed algorithms in an *asynchronous* network model. The asynchronous network is a point-to-point (store-and-forward) communication network, described by a *communication graph* $G = (V, E)$, of order $|V|$ and size $|E|$, where the set nodes V represents a set of state machines called the processors of the network and the set of links E represents communication channels operating between them. Let x and y be two nodes in V. There is a link from x

[*] Visiting the CINVESTAV under contract Marina-CONACyT 2002C013199A

to y if and only if x can talk directly to y. In our simplified model, we ignore all radio propagation effects and make all nodes have the same transmision range which we set to be equal to 1. The graph G is called the *unit distance graph*: there is a link between any two nodes whose distance is less than or equal to 1. The assumptions we make are that: a given MAC protocol is already in operation on the underlying network, all nodes know their geographic location, each node knows its immediate one-hop neighbors (those within its radio range) and, finally, we assume that all nodes but one, have a west-ward (north-ward) neighbor. Any node may suffer a crash failure and will be considered permanently out of service from then on.

We usually denote the ith processor in the system by p_i. FIFO queues are used to model asynchronous delivery of messages. A communication link is either unidirectional or bidirectional. A unidirectional communication link from process p_i to p_j transfers messages from p_i to p_j. The abstraction for such a unidirectional link is a first-in-first-out queue q_{ij}, that contains all messages sent by p_i to its neighbor p_j, that have not yet been received. The bidirectional communication link between p_i and p_j is modeled by two FIFO queues, one from p_i to p_j and other from p_j to p_i.

A *system configuration* c is a full description of a distributed system, at a particular time and consists of the state of every processor and the content of every queue. We use the term *step* for a computation step and we denote it by a. Let c_1 and c_2 be two configurations of the system, where c_2 is reached from c_1 by a single step a of a processor; we denote this fact by $c_1 \xrightarrow{a} c_2$, also we say that a is *applicable* to c_1. An *execution* $(c_1, a_1, c_2, a_2, \ldots)$ is an alternating sequence of configurations and steps such that $c_{i-1} \xrightarrow{a_{i-1}} c_i$.

No common memory is shared by the node's processors, and each node p_i has a distinct *identity* i. Each node processes messages received from its neighbors, performs local computations, and sends messages to its neighbors. All these actions are assumed to be performed in negligible time. All of the messages have a fixed length and may carry only a bounded amount of information. Each message sent by a node to its neighbors arrives within some finite but unpredictable time, unless a message lost happens. To model such events we extend the definition of a step to include environment steps of type $\text{loss}_{ij}(m)$ that is applicable to a configuration c_k in wich q_{ij} contains the message m. This step results in a configuration c_{k+1} in which m is removed from q_{ij}.

A *fair* execution is an execution in which, if infinitely often a processor has a step to execute then the processor executes this step infinitely often. Also, we do require that if a message is sent infinitely often, the message is received infinitely often. To satisfy fairness the receive step must be executed infinitely often while the loss step should not be executed infinitely often.

A *self-stabilizing* system can be started in an arbitrary configuration and will eventually exhibit a desired "legal", behavior. We define this legal behavior by a set of legal executions denoted LE. Every system execution of a self-stabilizing system should have a suffix that appears in LE. A configuration c is *safe* with regard to a task LE and an algorithm if every fair execution of the algorithm that starts from c belongs to LE. An algorithm is self-stabilizing for a task LE if every fair execution of the algorithm reaches a safe configuration with relation to LE [3].

The following complexity measures are used to evaluate performance of distributed algorithms operating on the above network. The *communication complexity* is the total number of messages sent during execution of the algorithm. The *time complexity* is the maximum time passed from its start to its termination, assuming that the time of delivering a message over each link is at most one unit of time. This bounded delay is assumed only for evaluating time complexity.

3 The Algorithm

This section is intended to sketch, from a formal view, the correctness of our setup and routing algorithms. It also fixes some bounds on the complexity of the whole process.

We claim that, once the process attains a steady condition, a distributed spanning tree is built on the set of participating nodes. The *ancestor*, *descendant*, and *level* concepts furtherly required are defined recursively: the root is axiomatically considered to be its own father, with its level equal to 0. A node's ancestor is said to be its father or any ancestor of its father. A node's level is said to be 1 plus the level of its father. A node's descendant is its sibling or a sibling of any descendant.

A START message indicates that the node must (re)trigger its father-searching procedure. During operations, nodes exchange three different messages: HELLO, DESCENDANTS and ANCESTORS. The HELLO message is issued by a node that has selected a place from its westward neighbors to be its father on the structure under construction. The receiver considers the sender to be its direct sibling, and gets prepared for further information coming from this place. The DESCENDANTS message is issued by a node to inform its father about the list, called Upper, of siblings (either direct or not) that can be reached through it. Finally, the ANCESTORS message is issued by a father to let its direct siblings know the list of nodes, called Lower, on the path from the resulting root to it.

We said that this algorithm evolves in cycles, each of them having two phases: during the first one, the convergecast, information flows from the leaves to the root, by means of the HELLO and DESCENDANTS messages. In the second phase, the broadcast, information goes from the root to the leaves, by means of the ANCESTORS messages. Each node has two timers that alternatively work to mark the ending of the corresponding phase. Upon the expiration of a timer the node is compelled to finish the proper flow with the information so far collected (See fig. 1).

Lemma 1. *There is exactly one node s, that becomes the root.*

Proof. Node s exists since there is, at least, one west-most node. i) if there is exactly one such place, then node s does not have any neighbor j, with relative polar coordinates (r_{js}, θ_{js}), for any $r_{js} > 0$ and $\theta_{js} \in (\frac{\pi}{2}, \frac{3\pi}{2}]$. Then, according

Fig. 1. Construction algorithm in node i

```
< 1> upon the reception of START
< 2>     select $j \in \mathrm{Neighbors}_i$, such that
< 3>     $j$ lies on the plane centered in $i$
< 4>     and has relative position $(r_{ji}, \theta_{ji})$,
< 5>     $r_{ji} > 0$ and $\theta_{ji} \in (\frac{\pi}{2}, \frac{3\pi}{2}]$;
< 6>     if does not exist such $j$
< 7>     then i am the root;
< 8>     else $\mathrm{father} = j$;
< 9>         send HELLO to $\mathrm{father}$
<10>         start $\mathrm{timer}_1$

<11> upon the reception of HELLO from $j$
<12>     $\mathrm{Sons}_i = \mathrm{Sons}_i \cup \{j\}$;
<13>     $\mathrm{ack}_j = F$;

<14> upon the expiration of $\mathrm{timer}_1$
<15>     for each $k \in \mathrm{Sons}_i : \mathrm{ack}_k == F$
<16>         $\mathrm{Sons}_i = \mathrm{Sons}_i \setminus \{k\}$;
<17>         $\mathrm{Upper}_i = \mathrm{Upper}_i \setminus \mathrm{Upper}_k$;
<18>         cancel $\mathrm{ack}_k$;
<19>     if i am the root
<20>     then $\mathrm{Lower}_i = \mathrm{Lower}_i \cup \{i\}$;
<21>         for any $k \in \mathrm{Sons}_i$
<22>             send ANCESTORS with $\mathrm{Lower}_i$ to $k$;
<23>     else $\mathrm{Upper}_i = \mathrm{Upper}_i \cup \{i\}$;
<24>         send DESCENDANTS with $\mathrm{Upper}_i$ to $\mathrm{father}$;
<25>         start $\mathrm{timer}_2$
<26>         for any $k \in \mathrm{Sons}_i$
<27>             $\mathrm{ack}_j = F$

<28> upon the expiration of $\mathrm{timer}_2$
<29>     $\mathrm{Lower}_i = \{i\}$;
<30>     for any $k \in \mathrm{Sons}_i$
<31>         send ANCESTORS with $\mathrm{Lower}_i$ to $k$;
<32>         $\mathrm{ack}_j = F$
<33>     goto <2>;

<34> upon the reception of DESCENDANTS from $j$
<35>     if $j \in \mathrm{Sons}_i$
<36>     then $\mathrm{Upper}_i = \mathrm{Upper}_i \cup \mathrm{Upper}_j$;
<37>         $\mathrm{ack}_j = T$;
<38>         if $\neg\exists k \in \mathrm{Sons}_i : \mathrm{ack}_k == F$
<39>         then stop $\mathrm{timer}_1$;
<40>             goto <19>;

<41> upon the reception of ANCESTORS from $j$
<42>     if $j == \mathrm{father}$ and $\mathrm{timer}_1$ is off
<43>     then stop $\mathrm{timer}_2$;
<44>         start $\mathrm{timer}_1$;
<45>         $\mathrm{Lower}_i = \mathrm{Lower}_j$;
<46>         goto <20>;
```

to the construction rule, it becomes the root. ii) assume there is more than one node having the same west-most (horizontal) coordinate, then all of them but the south-most will have at least one neighbor with relative polar coordinates within the selection range. Therefore, the south-most is the only place with conditions to become the root (see lines $< 1 > \ldots < 10 >$ of fig. 1).

Lemma 2. *There exists one single path from any node to the root.*

Proof. We proof by induction on the levels of the resulting structure. As usual, the root has level 0 and knows exactly one way to itself. Assume that our statement is true for all nodes up to level $n > 0$, therefore any node of level $n + 1$ only knows one way to the root that passes through its single ancestor of level n, i.e. its father.

Lemma 3. *Upon the termination of the first cycle, a distributed spanning tree is built on the set of active nodes and each one knows the list of lower and upper places that are reachable through its father and siblings, respectively.*

Proof. With the exception of the root, each node selects exactly one link (going to its father). Therefore, we have $|V| - 1$ links that make up the resulting structure.

We state that for any node j, if every $k \in \text{Lower}_j$ is active, then $s \in \text{Lower}_j$ and $j \in \text{Upper}_s$, as consequences of the converge and broadcast phases, respectively. Suppose this condition is true for any two nodes i and i' of level n and m, respectively. Also, suppose that i needs to reach i'. If $m < n$, then either $i' \in \text{Lower}_i$ or there exists exactly one path from i to i' that passes through the root s. On the other hand, if $m \geq n$ then, unless $i' \in \text{Upper}_i$, it is also granted that there exists exactly one path from i to j that passes through the root s.

We have shown that it is possible to find exactly one path between any two nodes, and also that the resulting graph has a minimum number of links (see lines $< 11 > \ldots < 13 >$ and $< 34 > \ldots < 46 >$ of fig. 1).

Lemma 4. *A node that loses the link to its father, eventually selects a new one and resynchronizes its subtree cycle with the rest of the structure.*

Proof. Suppose a node i has a direct sibling j, which in turn has a direct sibling k. Upon the event of a failure in j, two links will be dismissed: (i, j) and (j, k). Nevertheless, only k will be in charge of the recovery procedure, selecting a new father to reconnect its subtree. Recovery starts when k sends a HELLO message to its new father. This means that the timer 2 of the issuing node k has elapsed and it takes for granted that its former father is lost. Therefore, k starts its convergecast phase. In due time, it gathers all of the information about the upper nodes it is able to reach and now sends a DESCENDANTS message to its new father j'. Now k starts timer 2 again and sits down waiting for an ANCESTORS message coming from j'. At the other end of the new link, j' may be either in the convergecast or in the broadcast phase. In the first case, j' was just waiting for the DESCENDANTS message to update its $\text{Upper}_{j'}$ list. In due time, it will send an ANCESTORS message back to k wich will help it to update its Lower$_k$ list. In the second case, j' sends an ANCESTORS message back to k as in the previous case, but j' will not be able to update its own information up to the next cycle (see lines $< 28 > \ldots < 33 >$ of fig. 1).

We claim that the task ST of legitimate sequences is defined as the set of all configurations in which every configuration encodes a spanning tree of G. The preceeding lemma grants that it takes 2 cycles, at most, in order to reach a safe configuration that codifies a new tree.

Lemma 5. *A node with a timer that prematurely expires eventually updates with the correct structural information and resynchronizes its subtree cycle with the rest of the structure.*

Proof. Suppose timer 1 ends at node i and later a DESCENDANTS message is received at i from a direct sibling j. In this case, j has already been dismissed and the message is not accepted. Eventually, j will not receive the corresponding ANCESTORS message and will take for granted that its father is lost. This will trigger its recovery procedure as in the preceeding lemma(see lines $< 14 > \ldots < 27 >$ of fig. 1).

Suppose timer 2 ends at node i and later an ANCESTORS message is received a i from its father. In this case, as soon as timer 2 finishes, i starts its recovery procedure and selects a new father, therefore in the case of a late message from the former father, it will not be accepted(see lines $< 28 > \ldots < 33 >$ of fig. 1).

Lemma 6. *A cycle has message complexity $O(|V|)$ and time complexity $O(|V|)$, while a recovery takes an overhead $O(1)$.*

Proof. During the first cycle, a HELLO message is sent over each of the links that will make up the resulting tree. Next, at the end of the convergecast, a DESCENDANTS message will climb the same links to the root. Finally, during broadcast, an ANCESTORS message will flow down the leaves exactly on the same links but in opposite direction. From then on, only DESCENDANTS and ANCESTORS will traverse the resulting structure, unless an active node starts the recovery procedure sending, for one single time, a HELLO message to its new father. The rest of the synchronization can be regarded as being part of an ordinary cycle (see lines $< 1 > \ldots < 10 >$ of fig. 1).

So far, it has been shown that is possible to built up a spanning tree that keeps updated its structure information despite of link or node failures. Our construction is based on the local knowledge each node has about the position of its neighbors. Each node, except the resulting root, selects a neighbor within a geometric range. Clearly, this range can be reoriented, i.e. we can construct an north-ward tree, instead of a west-ward. But, what if we construct both trees at the same time? We will have two overlaying, self-reconfigurable, structures to profit from, as in a railroad system. It may have a great impact on routing procedures. Suppose we have an east-west tree T_{EW}, and a south-north tree T_{SN} on the same underlaying network. If node i requires to reach node i', it is granted that there are at two routes, one on each tree, and it is possible to install a performance metric to choose the best one. But also, in case of local problems, it could be possible to take an alternative path around the affected area. We present a very simple routing algorithm based on this possibility.

Let us assume again, that node i has an application message m, for node $d = i'$, then it encapsulates both data inside a routing packet that we will call INFO. Each place on the way from source to destination will obey the following procedure (See fig. 2).

Fig. 2. Routing algorithm in node i

```
upon the reception of INFO, carrying $m$ to $d$
    for some $T$ and $T'$, either $T_{NS}$ or $T_{EW}$
    if $i == d$
    then receive $m$;
    else
    if $\exists k \in \mathrm{Sons}_i^T$, $d \in \mathrm{Upper}_k^T$
    then forward INFO to $k$;
    else
    if $d \in \mathrm{Lower}_i^T$
    then forward INFO to father of $T$;
    else
        let $T'$ be the tree with the nearest root
        if $i$ is not the root of $T'$ and father is up
        then forward INFO to father of $T'$;
        else stand by until recovery;
```

Lemma 7. *Unless d crashes, m eventually arrives to d.*

Proof. It is a structural property of the overlaying trees (lemma 3). In this case each node must look at the two trees it belongs to, in order to find a direct route to d, or forwarding m to the closest root r' (either of T_{EW} or T_{SN}) where it is granted to exist a direct branch from r' to d (Nevertheless, it can find a better route on its way to r'). Finally, if m arrives to r' and there is not a built up branch going to d, it means that there has been a failure and it is better to wait until recovery is finished. Once the system is recovered, r' will forward m towards d, unless d is lost.

We leave for the next section some experimental evidence about the diameters of the resulting trees, which bound the complexity of the routing procedure.

4 Experimental Results

For each graph here considered we run 4 different families of algorithms, defined according to their tree construction procedures. Each family is said to perform an east-west scanning which produces a so-called horizontal (H) tree, next a similar procedure is performed according to a south-north scanning which produces a vertical (V) tree.

Basically all of the construction rules can be explained under the same principle, for an east-west (south-north respectively) scanning, take any node on a graph and select all of its incident edges going to the west (respectively north) and dismiss them all but one. Differences arise from the way this surviving edge is chosen:

Fig. 3. H tree produced with A1

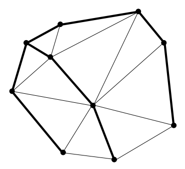


Fig. 4. V tree produced with A1

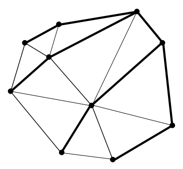


Fig. 5. H tree produced with A2

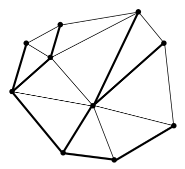
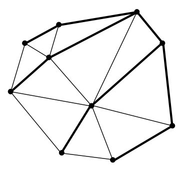


Fig. 6. V tree produced with A2



algorithm 1 (A1) for each node, we consider the west-ward neighbors' edges and dismiss all but the first edge in counterclockwise sense (fig. 3). Next we

perform the same operation over the 90 degrees rotated graph or, equivalently, from each node considering its north-ward neighbors' edges(fig. 4).

algorithm 2 (A2) for each node, select all of its incident edges going to the west and dismiss them all but the first on a clockwise sense (fig. 5). In contrast, for a vertical scanning, select all of the incident edges going to the north and dismiss them all but the first on a counterclockwise sense (fig. 6).

Fig. 7. H tree produced with A3

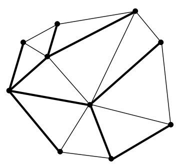


Fig. 8. V tree produced with A3

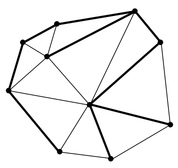


Fig. 9. H tree produced with A4

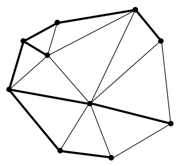
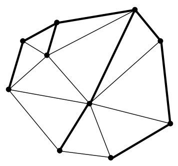


Fig. 10. V tree produced with A4



algorithm 3 (A3) again, in the horizontal scanning, select all of the incident edges going to the west and dismiss them all but one which is randomly picked up(fig. 7). In contrast, for a vertical scanning, select all the incident edges going to the north and dismiss them all but one which, as could be expected, is also randomly picked up(fig. 8).

algorithm 4 (A4) for each node, consider all of its incident edges going to the west and dismiss them all but the one closest to the horizontal axis. Next, we perform the same operation over the 90 degrees rotated graph or, equivalently, we dismiss all of the edges going to the north but the one closest to the vertical axis(fig. 10).

Three measures will be considered as a common basis in order to evaluate and compare the algorithms under study: i) HV correlation measures the number of edges belonging to the horizontal tree that also appear in the second one (vertical), ii) the mean diameter measures the mean longest path on the resulting trees, iii) finally, the degree and max. degree frequencies show the statistics of the nodes on the resulting constructions.

Table 1. 95% confidence intervals for the average correlation between H and V trees

	HV %correlation
A1	15.6 ± 0.2
A2	41.4 ± 0.2
A3	43.4 ± 0.3
A4	6.5 ± 0.1

Table 1 presents the HV correlation measured on each couple of trees constructed according to the 4 algorithms which were ran on 100-nodes graphs. Notice that in algorithm 1, the surviving edge is the last-one in clockwise sense, either on EW or SN scanning. In algorithm 2, the surviving edge is the first-one in clockwise (counterclockwise) sense for EW (SN) scanning. The poor performance on the HV correlation between the resulting trees is due to this selection rule, for it happens frequently that SN-surviving edge of one node is the same EW-surviving edge of a second node. In algorithm 3, the surviving edge is randomly selected in both scanningsm which does not seem to be a good rule, at least for the HV measure. In algorithm 4, the surviving edge is the one closest to the horizontal (vertical) axis for an east-west (south-north) scanning. This selection produces the best (lowest) correlation in all of the cases.

Table 2. 95% confidence intervals for mean diameters

order	20	40	60	80	100
A1	9.2 ± 0.06	14 ± 0.09	18 ± 0.11	21 ± 0.09	23 ± 0.16
A2	9.1 ± 0.07	14 ± 0.05	18 ± 0.20	21 ± 0.20	23 ± 0.29
A3	11 ± 0.07	17 ± 0.08	21 ± 0.15	21 ± 0.59	13 ± 0.92
A4	9.1 ± 0.07	14 ± 0.08	18 ± 0.10	21 ± 0.13	22 ± 0.42

Table 2 shows how the mean diameter evolves according to the graph order. The most interesting result is the way diameter grows in algorithm 3, which is completely different from the rest of the algorithms that have a smooth growing rate that follows an empirical law of the form $cn^{1/3}\log(n)$.

Table 3. Algorithms A1 to A4, degree and max. degree frequencies

	1	2	3	4	5	6	7	8	9
A1 deg. freq.	0.31	0.47	0.17	0.036	0.0097	0.0032	0.00051	0.00013	0
A1 max. deg.	0	0	0	0.14	0.5	0.29	0.051	0.013	0
A2 deg. freq.	0.32	0.45	0.17	0.038	0.012	0.0033	0.00059	0.00012	0
A2 max. deg.	0	0	0	0.071	0.56	0.29	0.059	0.012	0
A3 deg. freq.	0.35	0.4	0.2	0.049	0.0088	0.0013	0.00012	0	0
A3 max. deg.	0	0	0	0.31	0.55	0.13	0.012	0	0
A4 deg. freq.	0.32	0.46	0.17	0.039	0.011	0.0029	0.00073	0.00018	0
A4 max. deg.	0	0	0	0.091	0.55	0.27	0.073	0.018	0

Tables 3 shows the degree distribution for trees built up according to the algorithms under study, all of them ran on 100-nodes graphs. As for degree distribution, all of the tables show similar results which could be considered as a regular feature of the trees obtained according to this scanning principles. As for max. degree distribution, the slight differences arising in algorithm 3 could be the explanation for the results in table 2.

5 Conclusions and Further Work

Position is a source of global knowledge that adhoc wireless networks can profit from. Perhaps it might not be necessary to have a GPS on each node and it will do with two beacons located in orthogonal positions to let each place evaluate its coordinates based on field intensity.

We have introduced different ways to built up two overlaying spanning trees, all of them based on the same principles here presented. The impact of each procedure can be evaluated according to different parameters: mean diameter, mean degree and the number of common links between the resulting trees. We believe that the first two measures indirectly define the complexity of routing, while the last one indicates the robustness of the whole system.

About the experimental work, we should notice that having two trees built up from two different scanning senses makes the overlaying trees look like a subway or train system, this means that most of the nodes might work as an exchange points wich can have a great impact on the routing procedures, i.e. packets can travel over the horizontal tree and switch their routes to a vertical direction at any node, in order to shorten their path towards a given destination.

All of the procedures under test built up their corresponding trees from the local information that each node has about its neighbourhood. The importance of this approach becomes evident when we consider the following scenario: assume one of the nodes crashes, then only those neighbours directly connected with the missing place will be in charge of the reconfiguration. The rest of the nodes will not be (and do not need to be) aware of the changes triggered by the crash failure.

For further work, it would be interesting to find a tradeoff between timers' length (update rate) and communications overhead. Our solution might be applied in sensor networks too. In such case, this tradeoff would play a key role to lengthen battery lifetime [12],[6]. Also, we believe that the routing procedures here presented can be applied in mobile environments when all the nodes are moving in the same direction. Finally, we think that routing algorithms can be improved as long as we imposse restriction on the structural properties of the resulting trees.

References

1. K. Alzoubi, X.-Y. Li, Y. Wang, P.-J. Wan, O. Frieder, "Geometric Spanners for Wireless Ad Hoc Networks," IEEE Transactions on Parallel and Distributed Systems, Vol. 14, No. 5, May 2003.
2. P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. "Routing with guaranteed delivery in ad hoc wireless networks". In Proc. of Discrete Algorithms and Methods for Mobility (DIALM'99), pages 48–55, 1999.
3. S. Dolev, "Self-Stabilization", The MIT Press 2000.
4. L. Jia, R. Rajaraman, and C. Scheideler "On Local Algorithms for Topology Control and Routing in Ad hoc Networks," Proceedings of the 15th Annual ACM Symposium on Parallel Algorithms and Architectures, pages 220–229, June 2003.
5. E. Kranakis, H. Singh, and J. Urrutia, "Compass Routing on Geometric Networks", In Proceedings of 11th Canadian Conference on Computational Geometry, pp. 51–54, Vancouver, August, 1999.
6. X.-Y. Li, P.-J. Wan, Y. Wang, and O. Frieder, "Sparse power efficient topology for wireless networks, J. Parallel and Distributed Computing, to appear.
7. X.-Y. Li, G. Călinescu, P.-J. Wan, and Y. Wang, "Localized Delaunay Triangulation with Application in Ad Hoc Wireless Networks," IEEE Trans. on Par. Dist. Systems, 2003. To appear. (Modified version of the INFOCOM'2002 paper.)
8. X.-Y. Li, Y. Wang, O. Frieder, "Localized Routing for Wireless Ad Hoc Networks", in proceedings of IEEE ICC, 2003.
9. M. D. Penrose, "On k-Connectivity for a Geometric Random Graph", Random Structures and Algorithms, 15, 145-164, 1999.
10. M. D. Penrose, "The Longest Edge of the Random Minimal Spanning Tree", The Annals of Applied Probability, 7(2) 1997, 340-361.
11. R. Rajaraman, "Topology Control and Routing in Ad hoc Networks: A Survey," SIGACT News, 33:60–73, June 2002.
12. W.-Z. Song, Y. Wang, and X.-Y. Li, "Localized algorithms for energy efficient topology in wireless ad hoc networks," 5th ACM Int. Symp. on Mobile ad hoc Networking and Comp., Tokyo, Japan, 98–108, 2004.