Alcom-FT Technical Report Series
ALCOMFT-TR-03-170

Distributed Algorithms for Representative Problems in Ad-hoc Mobile Environments: A Critical Survey *

Sotiris Nikoletseas[†] Paul Spirakis[†]

Abstract

We examine here modeling and algorithmic issues arising in *mobile* computing environments, with an emphasis on the case where all devices can move. Having as a main goal to highlight *essential differences* (in models and algorithms correctness and efficiency) from classical *distributed* computing, we concentrate on three basic problems: leader election, point-to-point communication and reconfiguration of positions of devices. We discuss solutions to these problems that have appeared in the recent literature. In addition, we informally present and criticise some general algorithmic and modeling notions.

1 Introduction - Mobile vs Distributed Computing

The major technological advances in mobile networks have recently motivated the introduction of a *new* computing environment, called *mobile computing*.

Mobile computing is a new kind of distributed computing; several important features of it introduce essential modeling and abstraction differences. Among such features the most important is that some (or all) of the computing devices in the environment may move (in general, in arbitrary ways). In addition, usually the computing elements that are mobile, suffer from severe limitations with respect to:

(a) Communication: they can communicate up to a certain range

(b) Power (energy): They rely on a finite energy source (battery)

(c) Reliability: They are exposed to unexpected conditions while moving.

These limitations, when put together with the feature of motions, make the design of mobile information systems a quite hard task; communication among two devices may be broken due to distance while they move; the battery may fail in the middle of a global computing task; very fast motion may even introduce clock relativistic effects, thus making synchronism a hard task.

Hence, many distributed computing tasks, even the most elementary (leader election, counting, pair-wise communication etc.) have to be reconsidered in this new setting. As a consequence, new approaches are usually required.

An important special case of mobile computing environments is that in which all the devices are mobile (no fixed network exists). Such an environment is called an *ad-hoc* mobile computing environment. In contrast, a *hybrid* mobile computing environment allows for the coexistence of a fixed (backbone) network of stations that do not move and at the same time the existence of (usually many) moving devices.

*This work has been partially supported by the IST Programme of the European Union under contract numbers IST-1999-14186 (ALCOM-FT) and IST-2001-33116 (FLAGS).

[†]University of Patras and Computer Technology Institute. E-mails: nikole@cti.gr, spirakis@cti.gr.

1.1 Models for Mobile Algorithms

The design of high-level protocols or algorithms for mobile computing environments, requires the adoption of a suitable abstraction-model that precisely defines the new features. Any such model has two sides:

- *Local* i.e. the modeling of each computing device
- *Global* i.e. the modeling of motions and motion/communication territory and constraints

A local model of a mobile computing device, m, usually includes:

(a) A transmission radius, $R(t)$, at which m can send (or from which it can receive) messages. $R(t)$ may vary with time if the mobile device can adjust its power supply.

(b) A direction (or an angle) for communication at any position of the device; usually this is 360^o i.e. a broadcast is assumed.

(c) The amount $e(t)$ of energy still available at time t.

(d) The computing model of the device (i.e. an automaton, a machine with memory, a Turing Machine etc.)

Although the motion pattern of each device is a local feature, many models prefer to describe this in their global part, via a possible set of e.g. stochastic motions or in other summarizing ways.

Thus, the *global* model part of a mobile computing environment must characterize:

(a) The *motion space* i.e. the allowable motions (due to e.g. existence of obstacles or variable geometry)

(b) The form of motions allowed in that space (ideally, the instant velocity vector as a function of time)

(c) How energy is dissipated while moving.

Global models are either *explicit* (i.e. the motion space and patterns are given) or *implicit.*

Definition 1 An *implicit* global model for a mobile computing environment is usually a graph $G_t(V, E_t)$ where nodes are the devices and links *(time dependent)* represent direct communication capability due to instant proximity.

Usually, the form of G_t is assumed *not to be known* to the nodes (an *unknown* graph of communication).

Many research works assume G_t to be *quasi-static* i.e. they assume the existence of a time period Δ during which at most a certain number κ of links may appear or dissapear. Indeed, in some cases it is assumed that $\kappa = 1$.

Definition 2 The rate of change, h, of G_t is defined as $h = \kappa/\Delta$.

Notice that implicit representations *avoid* the geometry issues of the motion space; thus they are in essense combinatorial abstractions. *Intermediate* global models may represent the motion-allowable paths as a graph (the Motion Graph M). Notice that M does not (usually) change with time. Then, moving devices are seen as distinct particles, that occupy a node of M at each time and that move according to some law (e.g. from a vertex to some neighbour of it).

None of these abstractions should exclude *unnatural motions* (e.g. the instant jump from a point in space to another, not necessarly nearby. Examples here are quantum phenomena and telepresense).

So, we have:

Definition 3 A mobile computing model is a tuple $T = (\Gamma, E)$ where Γ is a global model and E a set of local models.

Definition 4 A problem specification P_i for the model T is a set of properties that T must satisfy.

Definition 5 A mobile computing protocol P, on T, for the problem Π, is a set of programs, P_I, for each device I, where each P_I has computing, communication, or motion commands, that satisfy the properties of Π when executed.

Note that we allow mobile protocols P to specify the motion of *some* of the devices to the benefit of the protocol. Such protocols are called semi-compulsory [12]. (Compulsory when the motions of *all* devices *obey* the protocol). Else, the protocol P is non-compulsory. The usefulness scenaria of semi-compulsory protocols include e.g. the case of a small set of moving devices that their purpose is to help in managing the mobile environment (this set is called the *support*, see [12]) or the case of some robots that move according to their program in order to carry out a global task (e.g. to coordinate, to rearrange, to achieve a formation, to help each other).

We should stress here that a model should allow (as a trivial case) some devices to stay still *(fixed stations)*. Fixed stations may be connected via permanent channels (*fixed* or *backbone* network) and may communicate in very different ways than the mobile devices (e.g. two fixed stations may be connected via a high-speed optical link while mobile devices usually are equipped only with digital radio capabilities).

We should also stress that explicit representation of geometry, in addition to the way local communication is modeled, introduces nice and deep problems; the graph G_t becomes e.g. an instant *geometric graph* (edges are where cycles of radius $R(t)$, centered at devices, intersect). Even *random* geometric graphs are extremely interesting (with respect to connectivity, coloring, diameter etc, see [24]). Failure patterns of individual devices may be modeled e.g. by a failure probability (or rate) per device and for a given (short) time, or for the whole computation.

The above modeling considerations necessarily introduce some important *parameters* for a mobile computing environment. Such parameters are e.g.:

- the communication radius $R(t)$
- the number of (mobile) devices
- the number of compulsory devices (support size)
- the rate of motion graph links updates h
- the motion space size (volume) $V(M)$ (e.g. the numbers of nodes of the graph representing M, or the actual volume)
- the total available energy $e(t)$

1.2 Some Basic Problems in Mobile Computing

While a mobile computing environment may execute complicated protocols for e.g. information system applications, however (as in the case of Distributed Computing) some *basic problems* are usually required to be faced by any protocol. Such problems, in distributed computing, are e.g.

- Leader Election
- Routing
- Communication of two stations a, b
- Counting the number of processes
- Critical regions/mutual exclusion, etc.

Note that (a) even the meaning of the problem may change in a mobile environment setting (b) new basic problems arise.

As an example, let us examine carefully the problem of Leader Election. Clearly, if two devices never arrive within communication range, the possibility of two "leaders" is apparent. Even if one requires a recognized leader device for a "connected" part of the communication graph, one then falls into the problem of specifying the duration and form of those "connected" parts.

There are at least two ways to make the problem meaningful here:

- Way 1. Given that a subset of devices $P \subseteq V$ of G_t is a *connected component* $\forall t \in I$ (where I is some time interval), the protocol should elect a unique leader in P, for times in I.
- Way 2. Let us assume that, after a finite number of topology changes (link updates) of G_t, G_t becomes a fixed G (*quiescent*). Then, it should eventually hold that: For each connected component, C, of G, there is a node $l \in V$ (the leader) whose identity is known to all nodes $i \in C$.

In many applications, it helps to ask for a second requirement in way 2 (a *routing information*): Eventually, each edge of G has a direction (imposed to it by its endpoint nodes) so that each connected component C of G becomes a *directed acyclic graph* with the leader as *the single* sink (such graphs are called leader-oriented graphs).

Note that the second way of specifying the problem is (perhaps) easier to handle mathematically. Note also that the requirement of quiescence after a "finite number" of link changes is semantically equivalent to the assumption that link changes *cease to happen* after some (long maybe) time. It is very tempting to us to stress here the similarity to on-line distributed algorithm. Indeed, any protocol that tries to *maintain* some global properties of G_t while links come and go, is *exactly* a distributed on-line algorithm. This remark opens the ground for efficiency measures of such protocols other than the usual, such as the competitive ratio, where the protocol P competes (continuously) against some (oblivious to P or not) "adversary" A that selects, at each time, which link to change.

Let us now carefully examine another problem: to implement the sending of a message from node a to node b. This is very easy in distributed computing (e.g. solved by flooding the network with the message from node a). However, the flooding way is not only energy non-efficient in a mobile setting, but also it may *fail* to solve the problem, simply because two parts of the network remain always disconnected.

Here, an assumption about motions is necessary for the problems *solvability*: it is necessary to assume that, given enough time, moving stations meet pairwise in such a way that a temporal path Π from a to b is formed. (A temporal path $\Pi = (a = i_0, i_1, i_2, \ldots, i_k = b)$ is a sequence of triples $(i_\lambda, i_{\lambda+1}, t_\lambda)$ meaning that i_λ and $i_{\lambda+1}$ communicate directly at time t_λ and $\forall \lambda$, $t_\lambda > t_{\lambda-1}$).

Indeed, nodes a, b may *never meet*, however the message may go via intermediate passes between pairs of nodes that meet.

Yet a problem of a different taste (that *does not have* a counterpart in classical distributed computing) is that of reconfiguring a "configuration" (i.e some initial placement of devices in M). This problem comes from the areas of Robotics (see [20]) or puzzles (e.g. the famous 16-puzzle or the Rubic cube) and it might be very hard even in its centralized version. Any way to solve a reconfiguration problem includes the subproblem of moving from some word of a finite group or semi-group to another, and the number of elementary motions (e.g. nearby station movements one at a time) is lower-bounded by the group diameter).

Distributed ways to solve such problems must specify carefully how the devices move and in which sequence. Not all motions may be possible due to e.g. geometry or robot shape restrictions;

also care is required so that e.g. the devices do not become disconnected and "lose their way" when they move.

In the sequel of the chapter we have selected three mobile computing algorithms (protocols) each solving one of the problems discussed above. We did that in order to provide the reader with an as wide as possible set of problems, solutions and modelling assumptions.

One of the protocols is due to our team, while the other two (and their analyses) are adapted from [38] and [47]. We do not insist on this being a complete set but rather a biased choice with the criteria of clarity, variability of model and approach and the fundamental nature of the problems.

1.3 Mathematical Tools for the Basic Problems

Each of the problems discussed above hints a different area of mathematics, mainly due to the modelling assumptions about motion. Thus, the implicit representation (via link changes) of motion in the leader-election problem recalls the use of combinatorics (graph theory) and some use of temporal logic and inductive inference.

Contrary, the explicit representation of motion in the basic communication problem, via independent *walks* (stochastic) that some devices do, leads to tools from the rich and recent theory of concurrent random walks and stochastic multi-particle interaction.

Finally, the reconfiguration problems hint either to group theory or to lattice distances; a flavour of algebraic combinatorics. We will see these in detail in the next chapters.

Let us note here that, in the problems of maintenance of properties of G_t (e.g. leaders, connectivity etc.) while links change, we believe that all the techniques of on-line algorithms can be re-used here (even statements of system "potential" changes).

To this end there is a relation with the approach of (stochastic) multi-particle interactions theory. That theory (originated in Statistical Physics) also examines the duration (or the eventual satisfaction) of global properties, and sometimes uses summary measures of the whole mobile system (e.g "heat"). There is a rich theory of phase transitions in multi-particle systems and quite similar phenomena are expected to appear in the system of many moving and locally interacting devices.

1.4 The Behavior of Mobile Computing Protocols

Many performance issues of mobile computing protocols (algorithms) are motivated by the Theory of Networks. We informally discuss some of them here:

(a) *Throughput.* It can be understood as the number of messages delivered to final destinations per unit time.

(b) *Scalability.* A protocol P is scalable when (1) its correctness is not affected when the network size changes and (2) its "performance" admits a smooth change when the size of the network changes. Here, the notion of performance can be quantified in various ways e.g. energy loss, time for a message to be finally delivered to destination, number of compulsory motions etc.

(c) *Stability.* The networks-motivated definition of stability is that the message population (number of messages originated but not yet delivered) in the system remains bounded as time grows. Although this definition still holds in mobile networks of devices, it does not capture entirely our intuition about a stable system. For example, an abnormal behaviour in link updates may cause certain protocol states to be repeated infinitely often, disallowing termination. The issue is quite open for research.

(d) *Fault-Tolerance* A mobile computing algorithm *tolerates k faults* when its correctness is not violated by up to k faults of devices during execution. A similar statement can be made for a protocol tolerating up to a certain *faults rate r* without correctness violation.

The semantics of faults are identical to those in Distributed Computing (random delays, permanent device failures, adversarial behavior of some mobile stations i.e. not following the protocol code etc.).

A quite interesting research aspect of mobile computing (with applications to mobile viruses in e.g. the Internet) is that of protocols of *pursuit-evation*: the mobile devices (or entities) here are separated into two groups with different objectives. The *fugitives* group wants to remain in the motion space as long as possible. The *hunters* group of devices has as its goal to "capture" or "find" the fugitives. The semantics of capturing a fugitive may vary from a simple meeting (or coexistence within some range) of the fugitive with one or more hunters, to a blocking of all possible moves of the fugitive by a set of hunters. Such protocols are motivated by the Graph Searching problem and have been proposed and analyzed in [1]. It should be clear that in pursuit-evation scenaria, the hunter devices are compulsory, since their motions are specified by the protocol. The motion *strategy* of fugitive devices must be assumed to be a *best response* to the hunters protocol (i.e. to be as adversarial as possible so that the task of the hunters is nontrivial). Clearly, game-theoretic notions can be applied in such settings.

1.5 Some More Algorithmic Problems in Mobile Environments

Some of the distinctive features of mobile computing are also the source of new problems. As an example, let us consider power-controlled networks, i.e. mobile networks where devices are able to change their transmission power (by controlling their battery according to the protocol). In such mobile environments, if a device u attempts to send a message with transmission power P_u, then all devices within range but with power less than αP_u ($\alpha > 0$ is a constant) cannot receive the message. Such mobile computing environments were considered in [2]. The authors of [2] discuss the complexity of routing in such environments, and propose near-optimal routing protocols for them.

Another algorithmic problem worthy of mentioning is that of devicing protocols for ad-hoc mobile networks that manage to implement group communication while ensuring a total order in the delivery of messages. There, motion makes preservation of delivery order hard (see e.g. [48]). A possible solution is to provide a subprotocol that causes a token to continually circulate through all the mobile devices. Such an idea is presented in [37], with a nice discussion about the performance of the protocol.

We remark here that the solvability and/or performance of many algorithmic problems in mobile computing depends (sometimes crucially) upon the assumed properties of the mobile devices (that are abstracted from technology). For example, the assumption of existence of GPS technology [25, 50, 51] makes the mobile devices position-aware with respect to a global system of coordinates. Position-awareness may trivialize some problems of reconfiguration.

2 An Algorithm for Leader Election

2.1 The Problem

The leader election problem is one of the most fundamental problems in distributed computing. Leader election is in fact a building block in the design of distributed protocols. For example, electing a unique node that generates a single token controlling the entrance to the critical section is essential to mutual exclusion algorithms. Further important uses of the leader election problem are the co-ordination in resourse sharing among a set of nodes towards achieving efficiency and fault-tolerance, and symmetry breaking (such as in a deadlock situation where a single node elected as a leader may undertake deadlock repairing actions). (See [39, 7] for a detailed study of leader election algorithms).

The leader election problem has many variants. For static networks, its most general version is, informally, the following: "Eventually, exactly one node in the network is designated a leader, while there should never be more than one leader during the execution of the leader election algorithm".

Leader election becomes especially important in a mobile setting, due to the frequent changes in the network topology because of the movement of hosts and other special reasons (permanent or temporary disconnections from the network, failures etc.). As an example, in a mutual exclusion algorithm, link/node failures may frequently result in token loss, thus it is crucial to elect a new leader for creating a new token. As an another example, consider a group communication algorithm for a mobile network (see e.g. [37]). In such a setting, mobility of hosts results in frequent group membership changes and a new leader must be elected among the new members of the group.

However, and mainly because of the dynamically changing network topology, solving the leader election problem in a mobile setting becomes more difficult and challenging compared to the static case. A usual complication is the possibility of disconnection of the ad-hoc network into connected components, or the merging of two or more connected components into one. In the first case it is essential to eventually elect a unique node in each connected component (note that after the disconnection and for a certain period, there might be components with no leader node). Similarly, in the case of two or more components that merge, there may be more than one leaders. In such a case, the algorithm must resolve the presence of multiple leaders by guaranteeing that only one leader survives.

In the light of the above, [38] proposes the following modified informal definition of the leader election problem in an ad-hoc mobile network: "Any connected component whose topology is static for sufficiently long time will eventually have exactly one leader". More formally, [38] provides the following problem statement for leader election in ad-hoc mobile networks.

Definition 6 [38] Let in each node i a variable lid_i that holds the identifier of the node currently considered the leader in the component of i. In every execution with a finite number of topology changes, eventually we must have that, for every connected component C of the topology graph, there is a node l in C such that $lid_i = l$ for all nodes i in C.

Note that the assumption in the above definition of "a finite number of topology changes" suggests (it is in fact equivalent) that the topology remains "for a sufficiently long time" unchanged.

2.2 The Model

The ad-hoc mobile networks considered in [38] are modeled as follows:

The network is comprised of a set of n independent mobile nodes, communicating in a wireless manner by message passing. The network is modeled as a dynamically changing, not necessarily connected, undirected graph, with vertices representing nodes and edges between vertices corresponding to nodes that are within each other's transmission range and can communicate.

The following assumptions on the mobile nodes and the network are made:

1. The nodes have unique node identities.

2. Communication links are bidirectional, reliable and FIFO. Unidirectional links, if any, are not used and ignored.

3. A protocol at the link layer ensures that each node knows the set of nodes with which it can currently directly (in one "hop") communicate (by providing indications of link formations and failures).

As far as the rate of topology changes is concerned, [38] distinguishes two cases:

a) The case of a single topology change: In this case it is assumed that only one change (either a link failure or a link formation) can occur at a time, in the sense that the next change occurs only after the network has been updated after the previous change.

b) The case of multiple concurrent topology changes.

Note that the latter case is much more difficult to cope with. [38] indeed proposes two different algorithms (one for each case). We disuss in the following section only the first one.

2.3 The Algorithm

The algorithm in [38] is in fact a modifcation of the TORA ([42]) routing algorithm, which in turn is based on a routing algorithm by Gafni and Bertsekas (GB, [28]).

Both GB and TORA follow the so-called "link reversal routing" paradigm (LRR, see [43] for an in-depth discussion).

The common characteristic of all LRR techniques is their goal to reduce (as much as possible) the routing overhead communication complexity due to topology changes. In LRR the reaction to topological changes is local, limited in frequency and scope of application. In contrast to trying to maintain at each node very powerful information (e.g. shortest paths) towards a destination node, LRR just maintains information necessary to support a directed acyclic graph (DAG), with the only sink being the destination node.

Maintaining a DAG guarantees the important property of loop freedom in the routing tables. Moreover, a DAG provides multiple, redundant paths to the destination node. However, a node in the DAG structure knows only its distance to its one-hop neighbors (i.e. those nodes within its transmission range) and is not aware of its distance to the destination node, i.e. no additive distance metric or topology composition information for a multi-hop propagation is supported.

Thus, LRR techniques may trade-off routing optimality for reduced topology adaptation communication overhead. Thus, LRR tends to be very adaptive and scalable. These properties are further supported by minimizing the frequency and scope of reactions to topology changes. In LRR, route maintenance (triggered by the failure of all outgoing links of a node) affects only the set of nodes whose directed paths (prior to failure) to the destination include the failed links. This is in sharp contrast to other approaches: in link-state approaches topological changes must be reported to all the nodes in the network, while in path finding techniques topology changes must be propagated to all nodes whose shortest path spanning trees are affected by the change.

Because of the above, it seems that LRR techniques are best suitable for networks of rather "medium" rate of topology changes i.e. when the mobility rate is not very slow to allow path maintenance or link- state techniques to be applicable, and not so fast to make any techniques trying to maintain some topological information not applicable, inefficient or even erroneous. For a rigorous analysis of the performance of link reversal techniques see [11]

In the next subsection we present the first ever LRR algorithm.

2.3.1 A Brief Description of the GB Algorithm

Gafni and Bertsekas introduce in [28] for the first time a Link Reversal Routing approach, by providing two algorithms for maintaining a destination-oriented Directed Acyclic Graph (DAG) in a network undergoing link failures.

The algorithms are called the "Partial Reversal Algorithm" and the "Full Reversal Algorithm". Both algorithms assign and dynamically update a unique "height" to each node, which is drawn from a totally ordered set of possible values. Each link between two nodes is directed from the node with the higher height to that with the lower height. The goal of the algorithm is to appropriately update the link directions (after the link failures) for maintaining a DAG in which the destination is the only sink.

In order for the algorithms to achieve this goal, whenever a node (other than the sink) loses all its outgoing links (i.e. when it loses its *last* outgoing or "downstream" link), either because of a link failure or because of a change in a neighbor's height, it calculates a new height for itself.

The two algorithms differ in the rule for calculating a new height. We here outline the Partial Reversal Algorithm, because it is this algorithm TORA and the algorithm of [38] are based upon. Furthermore, the Partial Reveral Algorithm tends to be the most efficient of the two, because the Full Reversal Algorithm (when a node has no outgoing links) reverses the direction of *all* of its links.

The height of a node i is a triple (α_i, β_i, i) of integers; the last component in the triple is the node's identity and is included in order to assure uniqueness. Indeed, assuming unique node identities the triples form a total order. Triples are compared lexicographically. In an abstract way, the triple of each node can be seen as its "height", where links between neighbor nodes are directed according to their relevant heights, i.e. from the higher to the lower. Thus, messages flow downstream from higher to lower height nodes. In fact, parameter α_i can been seen as a "base" or "reference" height level, with the other two parameters in the triple distinguishing heights of nodes with the same reference level.

A new iteration of the distributed algorithm is triggered whenever one or more nodes have no downstream links anymore. If node i loses all its outgoing (i.e. when it loses its last outgoing) links, it must compute a new height for itself. It chooses it to be (α_i', β_i', i), where α_i' is by one larger than the smallest α value among all its neighbors' heights. If i has a neighbor whose α height component is equal to α_i', then β_i' is set to be by one less than the smallest β value among all neighbors of i whose α height component equals α_i'. Otherwise the β component of i's height remains unchanged.

The above rule for setting α_i' ensures that node i will have at least one outgoing link, i.e., that (α_i', β_i', i) will be larger than the height of at least one neighbor, the one with the smallest height. The rule for setting β_i' aims at limiting the number of links incident on i that will reverse their direction, by keeping i's height smaller than that of any neighbors whose α height component is not smaller than α_i'. Reducing the number of link reversals contributes to the efficiency of the algorithm towards limiting the propagation of height changes (in contract to the Full Reversal Algorithm that reverses all link directions).

The GB algorithm is shown to be deadlock-free and loop-free. Furthermore, the maintenance of a DAG exhibits redundancy features by providing multiple routes to the destination. Assuming that the netowrk remains connected, the GB algorithm converges in finite time (i.e. after a finite number of iterations). When, however, the topological changes result in the disconnection of the network, the algorithm never converges by trapping in an infinite number of loops.

2.3.2 A Brief Description of the TORA Algorithm

Park and Corson [42] adapted the GB algorithm for routing in ad-hoc mobile networks, proposing TORA (Temporally Ordered Routing Algorithm). Their main contribution was a mechanism (maintaining a temporal marker) for distributed detection of network disconnections that result in the destination being no longer reachable. As mentioned in the previous section, the GB algorithms would cause an infinite cycle of messages in that case. TORA seems to be adaptive and scalable, thus it is possibly suitable for large ad-hoc mobile networks.

In TORA, the height of node i is a 5-tuple, $(\tau_i, oid_i, r_i, \delta_i, i)$. As in the GB algorithm, the last component is the node's identity, to ensure uniqueness. The first three components form a "reference" (or "base") level with the rest two differentiating node heights of the same reference level. A new reference level is started by node i if it loses all (i.e. its last) outgoing links due to a link failure. Then τ_i is set to the time when this occurs and oid_i is set to i, the originator node of this reference level. Component r_i in turn is used to modify the reference level. Initially, it is set to 0, the so-called "unreflected" reference level. It can be changed at some times to 1, indicating a "reflected" reference level, which is used in detecting network disconnections.

If the node i loses all its outgoing links due to a link failure, then the following route maintenance action is taken:

- *Case 1 (Generate new reference level):*

The δ_i, i components of the triple decide the link directions among all nodes with the same reference level so as to form a destination-oriented DAG. The originator of a new reference level assigns 0 to its δ component.

When a new reference level is created, say by node i, it is larger than any pre-existing reference level, since it is based on the current time. The originator notifies its neighbors of its new height. As shown in [38], this change eventually propagates among all nodes for whom i was on their only path to the destination. These are the nodes that must either form new paths to the destination or discover that, due to partitioning, there is no such path anymore.

Note that a node i can lose its last outgoing link due to a neighbor's height change under a variety of circumstances, which are explained in the sequel.

- *Case 2 (Propagate the highest neighbor's reference level):*

The first of these cases is when *not all* the neighbors of i have the same reference level. Then i sets its reference level to the largest among all its neighbors and sets its δ component to one less than the minimum δ value among all neighbors with the largest reference level (this partial reversal is similar to that in the GB algorithm).

- *Case 3 (Reflect back a higher sublevel):*

If all of i's neighbors do have the same reference level and it is an unreflected one, then i starts a reflection of this reference level by setting its reference level to the reflected version of its neighbors' (with $r_i = 1$) and its δ to 0.

- *Case 4 (Partition detected, deletion of invalid routes):*

If all of i's neighbors have the same reflected reference level with i as the originator, then i has detected a partition and takes appropriate action by deleting invalid routes.

- *Case 5 (Generate a new reference level):*

If all of i's neighbors have the same reference level with an originator other than i, then i starts a new reference level. This situation only happens if a link fails while the system is recovering from an earlier link failure.

2.3.3 Synopsis of the Leader Election Algorithm

As the authors state in [38], they made the following changes to TORA.

The height of each node i in their algorithm is now a 6-tuple $(lid_i, \tau_i, oid_i, r_i, \delta_i, i)$. The first component is the id of a node believed to be the leader of i's component. The remaining five components are the same as in TORA.

The reference level $(-1, -1, -1)$ is used by the leader of a component to ensure that it is a sink.

Remark that in TORA, once a network disconnection has been detected, the node that first detected the partition notifies the other nodes in its component so that they stop updating heights and sending (useless) messages. In the leader election algorithm, the node that detected the partition elects itself as the leader of the newly created component. It then transmits this information to its neighbors, who in turn propagate this information further to their neighbors and so on. Eventually all the nodes in the new component are notified of the leader change.

In the other case, i.e. when two or more components merge due to the activation of new links, the leader of the component whose identity is the smallest eventually becomes the only leader of the entire newly created component.

2.3.4 A Pseudo-code Description of the Algorithm

[38] describes the pseudo-code executed by node i, which we present below. Recall that each step is triggered either by the notification of the failure or formation of an incident link or by the receipt of a message from a neighbor.

Node i stores its neighbors' identities in a local variable N_i . When an incident link fails, i updates N_i . When an incident link is formed, i updates N_i and sends an Update message over that link with its current height. Upon receipt of an Update message, i updates a local data structure that keeps the current height reported for each of its neighbors. Node i uses this information to decide the direction of its incident links.

References in the pseudo-code below to variables $lid_j, \tau_j, oid_j, r_j, \delta_j$ for a neighbor node j of node i actually refer to the information that i has stored about j's height, in variable $height_i[j]$.

At the end of each step, if i's height has changed, then it sends an Update message with the new height to all its neighbors.

The pseudocode below explains how and when node i's height is changed. Parts B through D are executed only if the leader identity in the received Update message is the same as lid_i.

A. When node i has no outgoing links due to a link failure, then:
1. if node i has no incoming links as well then
2. $lid_i := i$
3. $(\tau_i, oid_i, r_i) := (-1, -1, -1)$
4. $\delta_i := 0$
5. else
6. $(\tau_i, oid_i, r_i) := (t, i, 0)$ (t is the current time)
7. $\delta_i := 0$

B. When node i has no outgoing links due to a link reversal following reception of an Update message and the reference levels (τ_j, oid_j, r_j) are not equal for all $j \in N_i$, then:
1. $(\tau_i, oid_i, r_i) := \max\{(\tau_j, oid_j, r_j) | j \in N_i\}$
2. $\delta_i := \min\{\delta_j | j \in N_i \text{ and } (\tau_j, oid_j, r_j) = (\tau_i, oid_i, r_i)\} - 1$

C. When node i has no outgoing links due to a link reversal following reception of an Update message and the reference levels (τ_j, oid_j, r_j) are equal with $r_j = 0$ for all $j \in N_i$, then:
1. $(\tau_i, oid_i, r_i) := (\tau_j, oid_j, 1)$ for any $j \in N_i$
2. $\delta_i := 0$

D. When node i has no outgoing links due to a link reversal following reception of an Update message and the reference levels (τ_j, oid_j, r_j) are equal with $r_j = 1$ for all $j \in N_i$ and $oid_j = i$, then:

1. $lid_i := i$
2. $(\tau_i, oid_i, r_i) := (-1, -1, -1)$
3. $\delta_i := 0$

E. When node i receives an Update message from neighboring node j such that $lid_j \neq lid_i$, then:
1. if $lid_i > lid_j$ or $(oid_i = lid_j$ and $r_i = 1)$ then
2. $lid_i := lid_j$
3. $(\tau_i, oid_i, r_i) := (0, 0, 0)$
4. $\delta_i := \delta_j + 1$

In part E, if the new identity is smaller than yours, then adopt it. If the new id is larger than yours, then adopt it, but only if it is the case that the originator of a new reference level has detected a partition and elected itself.

2.4 Analytic Considerations

We now present part of the correctness proof of [38] for their leader election algorithm.

The authors of [38] assume that each connected component is a leader-oriented DAG initially and that only one change (either a link failure or a link formation) can occur at a time. The next change occurs only after the entire network has recovered from the previous change (i.e. this is the case of single topology changes). They also assume that the system is synchronous, i.e., the execution occurs in rounds. Messages are sent at the beginning of each round and are received by the nodes to whom they were sent before the end of each round.

Theorem 1 [38] The algorithm ensures that each component eventually has exactly one unique leader.

Proof (sketch following [38]):

We consider the following three cases (the only ones causing changes):

Case 1: A link disappears at time t, causing node i to lose its last outgoing link but not disconnecting the component.

Case 2: A link appears at time t, joining two connected components.

Case 3: A link disappears at time t, causing node i to lose its last outgoing link and disconnecting the component.

In each case, [38] shows that eventually each component becomes a leader-oriented DAG:

Case 1: A link disappears at time t, causing node i to lose its last outgoing link but not disconnecting the component.

Let G be the directed graph representing the resulting component topology. Let l be the leader of the component. Then the component was an l-oriented DAG before the link was lost. Let V_l be the set of nodes that still have a path to l. At time t, the remaining nodes have a path to i; let this set be V_i. Let G_l be the graph induced by V_l, and G_i be the graph induced by V_i.

Definition 7 The frontier nodes of V_i are nodes that are adjacent to nodes in V_l. The edges between V_i and V_l are the frontier edges.

Let k be any node in V_i.

Definition 8 level(k) is the length of the longest path in G_i from k to i.

Note that level is defined with respect to the fixed G_i. Thus, even though the direction of edges changes as the algorithm proceeds, the levels remain the same.

Lemma 1 If k is on a path in G_i from a frontier node to i, then k's final height is $(l, t, i, 0, -level(k), k)$. Otherwise, k's final height is $(l, t, i, 1, -diff(k), k)$, where $diff(k) = \max\{level(h) | h \in V_i$ and k is reachable from h in $G_i\}$ - level(k).

Proof (sketch): By induction on the number of rounds r after t, [38] show that at the end of round r:

(a) If $r < level(k)$, then k's height is the same as it was at time t.

(b) If k is on a path from a frontier node to i and $r \geq level(k)$, then k's height is $(l, t, i, 0, -level(k), k)$.

(c) If k is not on a path from a frontier node to i and $level(k) \leq r < level(k) + 2 \cdot diff(k)$, then k's height is $(l, t, i, 0, -level(k), k)$.

(d) If k is not on a path from a frontier node to i and $r \geq level(k) + 2 \cdot diff(k)$, then k's height is $(l, t, i, 1, -diff(k), k)$. $\diamond$

Thus, Lemma 1 implies that the resulting graph is an l-oriented DAG, since all nodes in G_i now have paths to frontier nodes. The frontier edges are now directed from V_i to V_l because the τ-component in the heights of nodes in V_i is larger than for V_l (since the algorithm has access to synchronized or at least logical clocks).

Case 2: A link appears at time t, joining two formerly separate components C_1 and C_2 into a new component C.

Let l_1 be the leader of C_1 and l_2 the leader of C_2. Assume, without loss of generality, that $l_1 < l_2$. Suppose a link appears at time t between k_1 (a node in C_1) and k_2 (a node in C_2).

Lemma 2 Eventually l_1 becomes the leader of component C and C is an l_1-oriented DAG.

Proof (sketch): Let r be the number of rounds after t. At $r = 0$, k_1 and k_2 send Update messages to each other. Since k_1's leader l_1 is smaller than k_2's leader, l_2, k_2 updates its height to $(l_1, 0, 0, 0, \delta_{k_1+1}, k_2)$ and gets an outgoing link to k_1.

Let the value of $dist(k)$ for any node k in $partition_2$ be the shortest path distance from that node to node k_2 (the path distance is in terms of number of links). When $r < dist(k)$, the height of k remains unaffected since it has not yet received the Update message regarding the change in leadership. When $r = dist(k)$, k (including l_2) changes its height to $(l_1, 0, 0, 0, \delta_{k_1} + dist(k) + 1, k)$. Thus k now has a path to k_1 and its leader id has also changed to indicate a change in leadership. When $r > dist(k)$, the height of k remains unchanged.

Thus we see that when $r = dist(k)$, where k is the farthest node from k_2, all the nodes in $partition_2$ have updated their heights and have a route to k_1. The resulting graph (for the merged component) will be an l_1-oriented DAG, since k_1 is a node in $partition_1$ which is an l_1-oriented DAG. $\diamond$

Case 3: A link disappears at time t, causing node i to lose its last outgoing link and disconnecting the component.

The proof for case 3 is very similar to case 1, except that there will be no path from node i to a frontier node. The following condition will arise which is different from the conditions in case 1:

Let $r_1 = \max\{level(k) + 2 \cdot diff(k)\}$ for all k adjacent to i. At round r_1, the heights of all the adjacent nodes k will be $(l, t, i, 1, -diff(k), k)$ and node i will detect that a partition has occurred and will elect itself as the leader.

Lemma 3 At round r_1 a DAG with node i as the sink has already been formed.

Proof: We know from the proof of case 1 that, when $r > level(k) + 2 \cdot diff(k)$ for any node k other than i, node k has changed its height to $(l, t, i, 1, -diff(k), k)$ and has an outgoing link towards node i. This height of k will not change when $r > level(k) + 2 \cdot diff(k)$ and $r < r_1$. Also when $r = r_1 - 1$, one of the nodes k which is adjacent to i will change its height to $(l, t, i, 1, -diff(k), k)$ and have an outgoing link to node i. This node k will also be the last adjacent node of i to do so. ◇

Thus at r_1, when node i detects the partition, it changes its height to $(l, t, i, 1, -diff(k), k)$ and sends an Update message to its neighbors. This message is propagated throughout the new component. The resulting graph is an i-oriented DAG. The proof for this is the same as the proof for Lemma 2.

Thus we see from all the three cases that the algorithm will eventually ensure that each component has exactly one unique leader. Thus Theorem 1 has been proved.

3 A Communication Algorithm for High Mobility Networks

3.1 Summary

The work we present in this section focuses on networks with *high rate of mobility changes.* For such dynamically changing networks we propose protocols which exploit the *co-ordinated* (by the protocol) motion of a *small part* of the network. We show that such protocols *can be designed to* work correctly and efficiently even in the case of *arbitrary* (but not malicious) movements of the hosts not affected by the protocol.

We also propose a methodology for the *analysis of the expected* behavior of protocols for such networks, based on the *assumption* that mobile hosts (those whose motion is not guided by the protocol) conduct concurrent *random walks* in their motion space.

In particular, our work examines the fundamental problem of communication and proposes distributed algorithms for it. We provide *rigorous proofs* of their correctness, and also give performance analyses by combinatorial tools. Finally, we have evaluated these protocols by large scale simulation experiments.

3.2 The Case of High Mobility Rate

Ad-hoc mobile networks have been modeled by most researchers by a set of independent mobile nodes communicating by message passing over a wireless network. The network is modeled as a dynamically changing, not necessarily connected, undirected graph, with nodes as vertices and edges (virtual links) between vertices corresponding to nodes that can currently communicate. Such a model is used, for example, in [38, 35, 43]. See also Section 3 of this Chapter.

However, the proof of correctness of algorithms presented under such settings requires *a bound on the virtual link changes* that can occur at a time. As [43] also states, in ad-hoc mobile networks the topological connectivity is subject to frequent, unpredictable changes (due to the motion of hosts). If the *rate* of topological change is very high, then *structured* algorithms (i.e. algorithms that try to maintain connectivity related data structures, such as shortest paths, DAGs etc.) *fail* *to react fast enough* and [43] suggests that, in such cases, the only viable alternative is flooding. If the rate is low ("quasi-static" networks) or medium, then adaptive algorithmic techniques can apply. In such settings many nice works have appeared like e.g. the work of [38] on leader election, the work of [2] on communication (which however examines only *static* transmission graphs) etc. See the discussion on Link Reversal Routing (LRR) in the previous section of this Chapter.

The most common way to establish communication in ad-hoc networks is to form paths of intermediate nodes that lie within one another's transmission range and can directly communicate with each other [33, 35, 44]. The mobile nodes act as hosts and routers at the same time in order to propagate packets along these paths. Indeed, this approach of exploiting pair wise communication is common in ad-hoc mobile networks that cover a relatively small space (i.e. with diameter which is small with respect to transmission range) or are dense (i.e. thousands of wireless nodes) where all locations are occupied by some hosts; broadcasting can be efficiently accomplished.

In wider area ad-hoc networks with less users, however, broadcasting is impractical: two distant peers will not be reached by any broadcast as users may not occupy all intermediate locations (i.e. the formation of a path is not feasible). Even if a valid path is established, single link "failures" happening when a small number of users that were part of the communication path move in a way such that they are no longer within transmission range of each other, will make this path invalid. Note also that the path established in this way may be very long, even in the case of connecting nearby hosts.

We conjecture that, in cases of high mobility rate and/or of low density of user spreading, one can even state an *impossibility result*: any algorithm that tries to maintain a global structure with respect to the temporary network will be erroneous if the mobility rate is faster than the rate of updates of the algorithm.

In contrast to all such methods, we try to avoid ideas based on paths finding and their maintenance. We envision networks with *highly dynamic movement* of the mobile users, where the idea of "maintenance" of a valid path is inconceivable (paths can become invalid immediately after they have been added to the routing tables). Our approach is to take advantage of the mobile hosts natural movement by exchanging information whenever mobile hosts meet incidentally. It is evident, however, that if the users are spread in remote areas and they do not move beyond these areas, there is no way for information to reach them, unless the protocol takes special care of such situations.

3.3 The Basic Idea

In the light of the above, we propose the idea of forcing a *small* subset of the deployed hosts *to move as per the needs of the protocol*. We call this set the *"support"* of the network. Assuming the availability of such hosts, the designer can use them suitably *by specifying their motion* in certain times that the algorithm dictates.

We wish to admit that the assumption of availability of such hosts for algorithms deviates from the "pure" definition of ad-hoc mobile networks. However, we feel that our approach opens up an area for distributed algorithms design for ad-hoc mobile networks of high mobility rate.

Our approach is motivated by two research directions of the past:

(a) The *"two tier principle"* [33], stated for mobile networks with a fixed subnetwork however, which says that any protocol should try to move communication and computation to the fixed part of the network. Our assumed set of hosts that are coordinated by the protocol *simulates* such a (skeleton) network; the difference is that the simulation actually constructs a coordinated *moving* set.

(b) A usual scenario that fits to the ad-hoc mobile model is the particular case of *rapid deployment* of mobile hosts, in an area where there is no underlying fixed infrastructure (either because it is impossible or very expensive to create such an infrastructure, or because it is not established yet, or because it has become temporarily unavailable i.e. destroyed or down).

In such a case of rapid deployment of a number of mobile hosts, it is possible to have a small team of fast moving and versatile vehicles, to implement the support. These vehicles can be cars, jeeps, motorcycles or helicopters. We interestingly note that this small team of fast moving vehicles can also be a collection of independently controlled mobile modules, i.e. robots. This specific approach is inspired by the recent paper of J. Walter, J. Welch and N. Amato. In their paper "Distributed Reconfiguration of Metamorphic Robot Chains" ([47]) the authors study the problem of motion co-ordination in distributed systems consisting of such robots, which can connect, disconnect and move around. The paper deals with metamorphic systems where (as is also the case in our approach) all modules are identical. See Section 5 of this Chapter for a detailed discussion of the paper [47]. Note in particular that the approach of having the support moving in a co-ordinated way, *i.e. as a chain of nodes*, has some similarities to [47].

Related Work. In a recent paper ([36]), Q. Li and D. Rus present a model which has some similarities to ours. The authors give an interesting, yet different, protocol to send messages, which forces *all the mobile hosts to slightly deviate (for a short period of time) from their predefined, deterministic routes, in order to propagate the messages.* Their protocol is, thus, *compulsory* for any host, and it works only for deterministic host routes. Moreover, their protocol considers the propagation of only one message (end to end) each time, in order to be correct. In contrast, our support scheme allows for simultaneous processing of many communication pairs. In their setting, they show optimality of message transmission times.

M. Adler and C. Scheideler ([2]) deal only with *static* transmission graphs i.e. the situation where the positions of the mobile hosts and the environment do not change. In [2] the authors pointed out that static graphs provide a starting point for the dynamic case. In our work, we consider the *dynamic case* (i.e. mobile hosts move *arbitrarily*) and in this sense we extend their work. As far as performance is concerned, their work provides time bounds for communication that are proportional to the diameter of the graph defined by random uniform speading of the hosts.

3.4 The Model

3.4.1 The Motion Space

The set of previous research that follows the approach of slowly-changing communication graphs, models hosts motions only implicitly, i.e. via the pre-assumed upper bound on the rate of virtual link changes. In contrast, we propose an *explicit* model of motions because it is apparent that the motions of the hosts are the cause of the fragility of the virtual links.

Thus we distinguish explicitly between (a) the *fixed* (for any algorithm) *space* of possible motions of the mobile hosts and (b) the kind of motions that the hosts *actually perform* inside this space. In the sequel we have decided to model the space of motions only combinatorially, i.e. as a graph. We however believe that future research will complement this effort by introducing geometry details into the model.

In particular, we abstract the environment where the stations move (in three-dimensional space with possible obstacles) by a *motion-graph* (i.e. we neglect the detailed geometric characteristics of the motion). In particular, we first assume that each mobile host has a transmission range represented by a sphere tr centered by itself. This means that any other host inside tr can receive any message broadcast by this host. We approximate this sphere by a cube tc with volume $\mathcal{V}(tc)$, where $\mathcal{V}(tc) \simeq \mathcal{V}(tr)$. The size of tc can be chosen in such a way that its volume $\mathcal{V}(tc)$ is the maximum that preserves $\mathcal{V}(tc) < \mathcal{V}(tr)$, and if a mobile host inside tc broadcasts a message, this message is received by any other host in tc. Given that the mobile hosts are moving in the space $\mathcal{S}$, $\mathcal{S}$ is divided into consecutive cubes of volume $\mathcal{V}(tc)$.

Definition 9 The motion graph $G(V, E)$, $(|V| = n, |E| = m)$, which corresponds to a quantization of $\mathcal{S}$ is constructed in the following way: a vertex $u \in G$ represents a cube of volume $\mathcal{V}(tc)$. An edge $(u, v) \in G$ if the corresponding cubes are adjacent.

The number of vertices n, actually approximates the ratio between the volume $\mathcal{V}(\mathcal{S})$ of space $\mathcal{S}$, and the space occupied by the transmission range of a mobile host $\mathcal{V}(tr)$. In the case where $\mathcal{V}(\mathcal{S}) \simeq \mathcal{V}(tr)$ (the transmission range of the hosts approximates the space where they are moving), then $n = 1$. Given the transmission range tr, n depends linearly on the volume of space $\mathcal{S}$ regardless of the choice of tc, and $n = O\left(\frac{V(\mathcal{S})}{V(tr)}\right)$. Let us call the ratio $\frac{V(\mathcal{S})}{V(tr)}$ by the term *relative motion space size* and denote it by ρ. Since the edges of G represent neighboring polyhedra each node is connected with a constant number of neighbors, which yields that $m = \Theta(n)$. In our example where tc is a cube, G has maximum degree of six and $m \leq 6n$.

Thus *motion graph* G is (usually) a *bounded degree graph* as it is derived from a regular graph of small degree by deleting parts of it corresponding to motion or communication obstacles. Let Δ be the maximum vertex degree of G.

3.4.2 The Motion of the Hosts - Adversaries

The motion that the hosts perform (we mean here the hosts that are not part of the support i.e. those whose motion is *not specified* by the distributed algorithm) is *an input* to any distributed algorithm.

(a) In the general case, we assume that the motions of such hosts are decided by an *oblivious adversary*: The adversary determines motion patterns in any possible way but independently of the part of the distributed algorithm that specifies motions of the support. In other words, we exclude the case where some of the hosts not in the support are deliberately trying to *maliciously affect* the protocol (e.g. to avoid the hosts in the support). This is a pragmatic assumption usually followed by applications. We call such motion adversaries the *restricted motion adversaries.*

(b) For purposes of studying efficiency of distributed algorithms for ad-hoc networks *on the average*, we examine the case where the motions of any host not affected by the algorithm are modeled by *concurrent and independent random walks.* In fact, the assumption that the mobile users move randomly, either according to uniformly distributed changes in their directions and velocities or according to the random waypoint mobility model by picking random destinations, has been used by other research (see e.g. [29, 32]).

We interestingly note here a fundamental result of graph theory, according to which dense graphs look like random graphs in many of their properties. This has been noticed for expander graphs at least by [6, 3] and is captured by the famous Szemeredi's Regularity Lemma [46] which says that in some sense most graphs can be approximated by random-looking graphs.

In analogy, we conjecture that any set of *dense (in number)* but arbitrary otherwise motions of many hosts in the motion space can be approximated (at least with respect to their meeting and hitting times statistics) by a set of concurrent dense *random walks.* But the meeting times statistics of the hosts essentially determine the virtual fragile links of any ad-hoc network. We thus believe that our suggestion for adoption of concurrent random walks as a model for input motions, is not only a tool for average case performance analysis but it might in fact approximate well any ad-hoc network of dense motions.

3.5 The Problem

A *basic communication problem*, in ad-hoc mobile networks, is to send information from some *sender* user, MH_S, to another designated *receiver* user, MH_R.

One way to solve this problem is the protocol of notifying every user that the sender MH_S meets (and providing *all the information to it*) hoping that some of them will eventually meet the receiver MH_R.

> Is there a more efficient technique (other than notifying every user that the sender meets, in hope that some of them will then eventually meet the receiver) that will effectively solve the basic communication problem without flooding the network and exhausting the battery and computational power of the hosts?

A protocol solving this important communication problem is *reliable* if it allows the sender to be notified about delivery of the information to the receiver.

Note that this problem is very important and in fact fundamental, in the sense that no distributed computing protocol can be implemented in ad-hoc mobile networks without solving this basic communication problem.

3.6 A Set of Algorithms

3.6.1 The Basic Algorithm

In simple terms, the protocol works as follows: The nodes of the support move fast enough so that they visit (in sufficiently short time) the entire motion graph. Their motion is accomplished in a distributed way via a *support motion subprotocol* P_1. When some node of the support is within communication range of a sender, an underlying *sensor subprotocol* P_2 notifies the sender that it may send its message(s).

The messages are then stored "somewhere within the support structure". When a receiver comes within communication range of a node of the support, the receiver is notified that a message is "waiting" for him and the message is then forwarded to the receiver.

The messages received by the support are propagated within the structure when two or more members of the support meet on the same site (or are within communication range). A synchronization subprotocol P_3 is used to dictate the way that the members of the support exchange information.

In a way, the support Σ plays the role of a (moving) skeleton subnetwork (whose structure is defined by the motion subprotocol P_1), through which all communication is routed. From the above description, it is clear that the size, k, and the shape of the support may affect performance.

Note that the proposed scheme does not require the propagation of messages through hosts that are not part of Σ, thus its security relies on the support's security and is not compromised by the participation in message communication of other mobile users. For a discussion of intrusion detection mechanisms for ad-hoc mobile networks see [49].

3.6.2 The SNAKE Algorithm

The main idea of the protocol proposed in [12, 14] is as follows. There is a set-up phase of the ad-hoc network, during which a predefined set, k, of hosts, become the nodes of the support. The members of the support perform a leader election by running a randomized breaking symmetry protocol in anonymous networks (see e.g. [31]). This is run once and imposes only an initial communication cost. The elected leader, denoted by MS_0, is used to co-ordinate the support topology and movement. Additionally, the leader assigns local names to the rest of the support members MS_1, MS_2, ..., MS_{k-1}.

The nodes of the support move in a coordinated way, always remaining pair wise adjacent (i.e., forming a chain of k nodes), so that they sweep (given some time) the entire motion graph. This encapsulates the *support motion subprotocol* P_1^S. Essentially the motion subprotocol P_1^S enforces the support to move as a "snake", with the head (the elected leader MS_0) doing a random walk on the motion graph and each of the other nodes MS_i executing the simple protocol "move where MS_{i-1} was before". More formally, the movement of Σ is defined as follows:

> Initially, MS_i, $\forall i \in \{0, 1, \ldots, k-1\}$, start from the same area-node of the motion graph. The direction of movement of the leader MS_0 is given by a memoryless operation that chooses *randomly the direction* of the next move. Before leaving the current area-node, MS_0 sends a message to MS_1 that states the new direction of movement. MS_1 will change its direction as per instructions of MS_0 and will propagate the message to MS_2. In analogy, MS_i will follow the orders of MS_{i-1} after transmitting the new directions to MS_{i+1}. Movement orders received by MS_i are positioned in a queue Q_i for sequential processing. The very first move of MS_i, $\forall i \in \{1, 2, \ldots, k-1\}$ is delayed by a δ period of time.

We assume that the mobile support hosts move with a common speed. Note that the above described motion subprotocol P_1^S enforces the support to move as a "snake", with the head (the elected leader MS_0) *doing a random walk on the motion graph* G and each of the other nodes MS_i executing the simple protocol "move where MS_{i-1} was before". This can be easily implemented because MS_i will move following the edge from which it received the message from MS_{i-1} and therefore our protocol does not require common sense of orientation.

The purpose of the random walk of the head is to ensure a *cover* (within some finite time) of the whole motion graph, without memory (other than local) of topology details. Note that this memoryless motion also ensures fairness. The value of the *random walk principle* of the motion of the support will be further justified in the correctness and the efficiency parts of the paper, where we wish our basic communication protocol to meet some performance requirements *regardless of the motion of the hosts not in* Σ.

A modification of M_Σ^S is that the head does a random walk on *a spanning subgraph* of G (e.g. a spanning tree). This modified M_Σ (call it M'_Σ) is more efficient in our setting since "edges" of G just represent adjacent locations and "nodes" are really possible host places.

A simple approach to implement the support *synchronization subprotocol* P_3^S is to use a forward mechanism that transmits incoming messages (in-transit) to neighboring members of the support (due to P_1^S at least one support host will be close enough to communicate). In this way, all incoming messages are eventually copied and stored in every node of the support. This is not the most efficient storage scheme and can be refined in various ways in order to reduce memory requirements.

3.6.3 The RUNNERS Algorithm

A different approach to implement M_Σ is to allow each member of Σ not to move in a snake-like fashion, but to perform an *independent* random walk on the motion graph G, i.e., the members of Σ can be viewed as "runners" running on G. In other words, instead of maintaining at all times pair wise adjacency between members of Σ, all hosts sweep the area by moving independently of each other. When two runners meet, they exchange any information given to them by senders encountered using a new synchronization subprotocol P_3^R. As in the snake case, the same underlying sensor sub-protocol P_2 is used to notify the sender that it may send its message(s) when being within communication range of a node of the support.

As described in [12], the runners protocol does not use the idea of a (moving) backbone subnetwork as no motion subprotocol P_1^R is used. However, all communication is still routed through the support Σ and we expect that the size k of the support (i.e., the number of runners) will affect performance in a more efficient way than that of the snake approach. This expectation stems from the fact that each host will meet each other in parallel, accelerating the spread of information (i.e., messages to be delivered).

A member of the support needs to store all undelivered messages, and maintain a list of receipts to be given to the originating senders. For simplicity, we can assume a generic storage scheme where all undelivered messages are members of a set S_1 and the list of receipts is stored on another set S_2. In fact, the unique ID of a message and its sender ID is all that is needed to be stored in S_2.

When two runners meet at the same vertex of the motion graph G, the synchronization subprotocol P_3^R is activated. This subprotocol imposes that when runners meet on the same vertex, their sets S_1 and S_2 are synchronized. In this way, a message delivered by some runner will be removed from the set S_1 of the rest of runners encountered, and similarly delivery receipts already given will be discarded from the set S_2 of the rest of runners.

The synchronization subprotocol P_3^R is partially based on the *two-phase commit* algorithm as presented in [39] and works as follows.

> Let the members of Σ residing on the same site (i.e., vertex) u of G be $MS_1^u, \ldots, MS_j^u$. Let also $S_1(i)$ (resp. $S_2(i)$) denote the S_1 (resp. S_2) set of runner MS_i^u, $1 \leq i \leq j$. The algorithm assumes that the underlying sensor sub-protocol P_2 informs all hosts about the runner with the lowest ID, i.e., the runner MS_1^u. P_3^R consists of two rounds.
>
> *Round* 1*:* All $MS_1^u, \ldots, MS_j^u$ residing on vertex u of G, send their S_1 and S_2 to runner MS_1^u. Runner MS_1^u collects all the sets and combines them with its own to compute its new sets S_1 and S_2: $S_2(1) = \bigcup_{1 \leq l \leq j} S_2(l)$ and $S_1(1) = \bigcup_{1 \leq l \leq j} S_1(l) - S_2(1)$.
>
> *Round* 2*:* Runner MS_1^u broadcasts its decision to all the other support member hosts. All hosts that received the broadcast apply the same rules, as MS_1^u did, to join their S_1 and S_2 with the values received. Any host that receives a message at Round 2 and which has not participated in Round 1, accepts the value received in that message as if it had participated in Round 1.

This simple algorithm guarantees that mobile hosts which remain connected (i.e., are able to exchange messages) for two continuous rounds, will manage to synchronize their S_1 and S_2. Furthermore, on the event that a new host arrives or another disconnects during Round 2, the execution of the protocol will not be affected. In the case where runner MS_1^u fails to broadcast in Round 2 (either because of an internal failure or because it has left the site), then the protocol is simply re-executed among the remaining runners.

Remark that the algorithm described above does not offer a mechanism to remove message receipts from S_2; eventually the memory of the hosts will be exhausted. A simple approach to solve this problem and effectively reduce the memory usage is to construct an order list of IDs contained in S_2, for each receiver. This ordered sequence of IDs will have gaps - some messages will still have to be confirmed, and thus are not in S_2. In this list, we can identify the maximum ID before the first gap. We are now able to remove from S_2 all message receipts with smaller ID than this ID.

3.7 Analysis of the SNAKE Algorithm

3.7.1 Preliminaries

In the sequel, we assume that the head of the snake does a *continuous time random walk* on $G(V, E)$, without loss of generality (if it is a discrete time random walk, all results transfer easily, see [5]). We define the random walk of a host on G that induces a continuous time Markov chain M_G as follows: The states of M_G are the vertices of G and they are finite. Let s_t denote the state of M_G at time t. Given that $s_t = u$, $u \in V$, the probability that $s_{t+dt} = v$, $v \in V$, is $p(u, v) \cdot dt$ where

$$p(u,v) = \begin{cases} \frac{1}{d(u)} & \text{if } (u,v) \in E \\ 0 & \text{otherwise} \end{cases}$$

and $d(u)$ is the degree of vertex u.

We assume that all random walks are *concurrent* and that there is a global time t, not necessarily known to the hosts.

Note 1 Since the motion graph G is finite and connected, the continuous Markov chain abstracting the random walk on it is automatically time-reversible.

Definition 10 $P_i(\mathrm{E})$ is the probability that the random walk satisfies an event E given it started at vertex i.

Definition 11 For a vertex j, let $T_j = min\{t \geq 0 : s_t = j\}$ be the first hitting time of the walk onto that vertex and let $E_i T_j$ be its expected value, given that the walk started at vertex i of G.

Definition 12 For the random walk of any particular host, let $\pi()$ be the stationary distribution of its position after a sufficiently long time.

We denote $E_\mu\,[\,]$ the expectation for the chain started at time 0 from any vertex with distribution μ (e.g. the initial distribution of the Markov chain).

We know (see [5]) that for every vertex σ, $\pi(\sigma) = \frac{d(\sigma)}{2m}$ where $d(\sigma)$ is the degree of σ in G and $m = |E|$.

Definition 13 Let $p_{j,k}$ be the transition probability of the random walk from vertex j to vertex k. Let $p_{j,k}(t)$ be the probability that the random walk started at j will be at $k \in V$ in time t.

Definition 14 Let $X(t)$ be the position of the random walk at time t.

3.7.2 Correctness Guarantees under the Restricted Motion Adversary

Let us now consider the case where any sender or receiver is allowed a general, unknown motion strategy, but its strategy is provided by a restricted motion adversary. This means that each host not in the support either executes a deterministic motion (which either stops at a node or cycles forever after some initial part) or it executes a stochastic strategy which however is *independent* of the motion of the support.

Theorem 2 The support Σ and the management subprotocol M_Σ guarantee reliable communication between any sender-receiver (MH_S, MH_R) pair in finite time, whose expected value is bounded only by a function of the relative motion space size ρ and does not depend on the number of hosts, and is also independent of how MH_S, MH_R move, provided that the mobile hosts not in the support do not deliberately try to avoid the support.

Proof: For the proof purposes, it is enough to show that a node of Σ will meet MH_S and MH_R infinitely often, with probability 1 (in fact our argument is a consequence of the Borel-Cantelli Lemmas for infinite sequences of trials). We will furthermore show that the first meeting time M (with MH_S or MH_R) has an expected value (where expectation is taken over the walk of Σ and any strategy of MH_S (or MH_R) and any starting position of MH_S (or MH_R) and Σ) which is bounded by a function of the size of the motion graph G only. This then proves the Theorem since it shows that MH_S (and MH_R) meet with the head of Σ infinitely often, each time within a bounded expected duration.

So, let EM be the expected time of the (first) meeting and $m^* = supEM$, where the supremum is taken over all starting positions of both Σ and MH_S (or MH_R) and all strategies of MH_S (one can repeat the argument with MH_R).

We proceed to show that we can construct for the walk of Σ's head a *strong stationary time sequence* V_i such that for all $\sigma \in V$ and for all times t

$$P_i(X(V_i) = \sigma \ \mid \ V_i = t) \ = \ \pi(\sigma)$$

Remark that strong stationary times were introduced by Aldous and Diaconis in [4].

Notice that at times V_i, MH_S (or MH_R) will necessarily be at some vertex σ of V, either still moving or stopped.

Let u be a time such that for X,

$$p_{j,k}(u) \geq \left(1 - \frac{1}{e}\right)\pi(k)$$

for all j, k. Such a u always exists because $p_{j,k}(t)$ converges to $\pi(k)$ from basic Markov Chain Theory.

Note that u depends only on the structure of the walk's graph, G. In fact, if one defines separation from stationarity to be $s(t) = max_j s_j(t)$ where $s_j(t) = sup\{s : p_{ij}(t) \ \geq \ (1 - s)\pi(j)\}$ then

$$\tau_1^{(1)} = min\{t : s(t) \ \leq \ e^{-1}\}$$

is called the *separation threshold time*. For general graphs G of n vertices this quantity is known to be $\mathcal{O}(n^3)$ ([8]).

Now consider a sequence of stopping times $U_i \in \{u, 2u, 3u, \dots\}$ such that

$$P_i(X(U_i) = \sigma \ \mid \ U_i = u) \ = \ \frac{1}{e}\,\pi(\sigma) \tag{1}$$

for any $\sigma \in V$. By induction on $\lambda \geq 1$ then

$$P_i(X(U_i) = \sigma \ \mid \ U_i = \lambda u) \ = \ e^{-(\lambda-1)}\frac{1}{e}\,\pi(\sigma)$$

This is because of the following: First remark that for $\lambda = 1$ we get the definition of U_i. Assume that the relation holds for $(\lambda - 1)$ i.e.

$$P_i(X(U_i) = \sigma \ \mid \ U_i = (\lambda-1)u) \ = \ e^{-(\lambda-2)}\frac{1}{e}\,\pi(\sigma)$$

for any $\sigma \in V$. Then $\forall \sigma \in V$

$$\begin{aligned} P_i(X(U_i) = \sigma \ \mid \ U_i = \lambda u) \ &= \ \sum_{\alpha \in V} P_i(X(U_i) = \alpha \ \mid \ U_i = (\lambda-1)u) \ \cdot \\ &\quad \cdot P(X(U_i + u) = \sigma \ \mid \ X(U_i) = \alpha) \\ &= \ e^{-(\lambda-2)}\,\frac{1}{e}\sum_{\alpha \in V}\pi(\alpha)\,\frac{1}{e}\,\pi(\sigma) \qquad \text{from (1)} \\ &= \ e^{-(\lambda-1)}\,\frac{1}{e}\,\pi(\sigma) \end{aligned}$$

which ends the induction step. Then, for all σ

$$P_i(X(U_i) = \sigma) \ = \ \pi(\sigma) \tag{2}$$

and, from the geometric distribution with $q \ = \ \frac{1}{e}$,

$$E_i U_i \ = \ u\left(1 \ - \ \frac{1}{e}\right)^{-1}$$

Now let $c \ = \ u\frac{e}{e \ - \ 1}$. So, we have constructed (by (2)) a *strong stationary time sequence* U_i with $EU_i = c$. Consider the sequence $0 = U_0 < U_1 < U_2 < \ldots$ such that for $i \geq 0$

$$E(U_{i+1} - U_i \ \mid \ U_j, \ \ j \leq i) \leq c$$

But, from our construction, the positions $X(U_i)$ are *independent* (of the distribution $\pi()$), and, in particular, $X(U_i)$ *are independent* of U_i.

Therefore, regardless of the strategy of MH_S (or MH_R) and because of the independence assumption, the support's head has chance at least $\min_\sigma \pi(\sigma)$ to meet MH_S (or MH_R) at time U_i, independently of how i varies.

So, the meeting time M satisfies $M \leq U_T$ where T is a stopping time with mean

$$ET \leq \frac{1}{min_\sigma \pi(\sigma)}$$

Note that the idea of a stopping time T such that $X(T)$ has distribution π and is independent of the starting position is central to the standard modern Theory of Harris - recurrent Markov Chains (see e.g. [26]).

From Wald's equation for the expectation of a sum of a random number of independent random variables (see e.g. [5, 45]) then

$$EU_T \ \leq \ c \cdot ET \quad \Rightarrow \quad m^* \ \leq \ c\,\frac{1}{\min_\sigma \pi(\sigma)}$$

Note that since G is produced as a subgraph of a regular graph of fixed degree Δ we have

$$\frac{1}{2m} \;\leq\; \pi(\sigma) \;\leq\; \frac{1}{n}$$

for all σ ($n = |V|$, $m = |E|$), thus

$$ET \;\leq\; 2m$$

hence

$$m^* \leq 2mc \;=\; \frac{e}{e-1}\, 2mu$$

Since m, u only depend on G, this proves the Theorem.

$\diamond$

Corollary 1 If Σ's head walks randomly in a *regular* spanning subgraph of G, then $m^* \leq 2cn$.

3.7.3 Worst Case Time Efficiency for the Restricted Motion Adversary

Clearly, one intuitively expects that if $k = |\Sigma|$ is the size of the support then the higher k is (with respect to n), the best the performance of Σ gets.

By working as in the construction of the proof of Theorem 2, we can create a sequence of strong stationary times U_i such that $X(U_i) \in F$ where F={σ: σ is a position of a host in the support}. Then $\pi(\sigma)$ is replaced by $\pi(F)$ which is just $\pi(F) = \sum \pi(\sigma)$ over all $\sigma \in F$. So now m^* is bounded as follows:

$$m^* \;\leq\; c\, \frac{1}{\min_{\sigma \in J}\left(\sum \pi(\sigma)\right)}$$

where J is any induced subgraph of the graph of the walk of Σ's head such that J is the neighborhood of a vertex σ of radius (maximum distance simple path) at most k. The quantity

$$\min_J \left(\sum_{\sigma \in J} \pi(\sigma) \right)$$

is then at least $\frac{k}{2m}$ (since each $\pi(\sigma)$ is at least $\frac{1}{2m}$ and there are at least k vertices σ in the induced subgraph), hence, $m^* \;\leq\; c\, \frac{2m}{k}$.

The overall communication is given by:

$$T_{total} \;=\; X \;+\; T_\Sigma \;+\; Y \tag{3}$$

where:

- X is the time for the sender node to reach a node of the support.
- T_Σ is the time for the message to propagate inside the support. Clearly, $T_\Sigma = \mathcal{O}(k)$, i.e. linear in the support size.
- Y is the time for the receiver to meet with a support node, after the propagation of the message inside the support.

We have for all MH_S, MH_R:

$$E(T_{total}) \;\leq\; \frac{2mc}{k} \;+\; \Theta(k) \;+\; \frac{2mc}{k}$$

The upper bound achieves a minimum when $k = \sqrt{2mc}$.

Lemma 4 For the walk of Σ's head on the entire motion graph G, the communication time's expected value is bounded above by $\Theta(\sqrt{mc})$ when the (optimal) support size $|\Sigma|$ is $\sqrt{2mc}$ and c is $\frac{e}{e-1}u$, u being the "separation threshold time" of the random walk on G.

Remark that other authors use for u the symbol $\tau_1^{(1)}$ for a symmetric Markov Chain of continuous time on n states.

3.7.4 Tighter Bounds for the Worst Case of Motion

To make our protocol more efficient, we now force the head of Σ to perform a random walk on a *regular spanning graph* of G. Let $G_R(V, E')$ be such a subgraph.

Our improved protocol versions assume that (a) such a subgraph exists in G and (b) is given in the beginning to all the stations of the support.

Then, for any $\sigma \in V$, and for this new walk X', we have for the steady state probabilities $\pi_{X'}(\sigma) = \frac{1}{n}$ for all σ (since in this case $d(\sigma) = r$ and $2m = rn$, for all σ, where r is the degree of any σ).

Let now M be again the first meeting time of MH_S (or MH_R) and Σ's head. By the stationarity of X' and since all $\pi_{X'}(\sigma)$ are equal,

$$\int_0^t P(MH_S, X' \text{ are together at time s})\, ds \;=\; \frac{t}{n} \tag{4}$$

Now let

$$p^*(t) \;=\; \max_{x,v} p_{x,v}(t)$$

Regardless of initial positions, the chance that MH_S (or MH_R), Σ's head are together (i.e. at the same vertex) at time u is *at most* $p^*(u)$. So, by assuming first that Σ's head starts with the stationary distribution, we get

$$P(MH_S, X' \text{ are together at time s}) = \int_0^s f(u) P(\text{together at time s} \mid M = u)\, du$$

where $f(u)$ is the (unknown) probability density function of the meeting time M. So

$$P(MH_S, X' \text{ together at time s}) \;\leq\; \int_0^s f(u) p^*(s-u)\; du$$

But we know (see [5]) that

Lemma 5 [5] There exists an absolute constant K such that

$$p^*(t) \;\leq\; \frac{1}{n} \;+\; Kt^{-\frac{1}{2}} \quad \text{where } 0 \leq t < \infty$$

Thus

$$P(MH_S, X' \text{ are together at time s}) \leq \frac{1}{n}P(M \leq s) \, + \, K\int_0^s f(u)(s-u)^{-\frac{1}{2}}du$$

So, from (4), we get

$$\begin{aligned} \frac{t}{n} \;&\leq\; \frac{1}{n}\int_0^t P(M \leq s)ds \;+\; K\int_0^t f(u)du \int_0^t (s-u)^{-\frac{1}{2}}ds \\ &=\; \frac{t}{n} \;-\; \frac{1}{n}\int_0^t P(M > s)ds \,+\, 2K\int_0^t f(u)(t-u)^{-\frac{1}{2}}du \\ &\leq\; \frac{t}{n} - \frac{1}{n}Emin(M,t) \;+\; 2Kt^{-\frac{1}{2}} \end{aligned}$$

So, the expected of the minimum of M and t is

$$Emin(M,t) \;\leq\; 2Knt^{-\frac{1}{2}}$$

Taking $t_0 = (4Kn)^2$, from Markov's inequality we get $P(M \leq t_0) \;\geq\; \frac{1}{2}$.

When Σ starts at some arbitrary vertex, we can use the notion of *separation time* $s(u)$ (a time to approach stationarity by at least a fraction of $(1 \;-\; \frac{1}{e})^{-1}$) to get

$$P(M \leq u + t_0) \;\geq\; \frac{1 \;-\; s(u)}{2}$$

i.e., by iteration,

$$EM \;\leq\; \frac{2(u \;+\; t_0)}{1 \;-\; s(u)}$$

where u is as in the construction of Theorem 2. Thus,

$$m^* \leq \frac{2}{(1 \;-\; \frac{1}{e})}(t_0 + u)$$

but for regular graphs it is known (see e.g. [5]) that $u = \mathcal{O}(n^2)$ implying (together with our remarks about Σ's size and set of positions) the following theorem:

Theorem 3 By having Σ's head to move on a regular spanning subgraph of G, there is an absolute constant $\gamma > 0$ such that the expected meeting time of MH_S (or MH_R) and Σ is bounded above by $\gamma\,\frac{n^2}{k}$.

Again, the total expected communication time is bounded above by $2\gamma\frac{n^2}{k} \;+\; \Theta(k)$ and by choosing $k = \sqrt{2\gamma n^2}$ we can get a best bound of $\Theta(n)$ for a support size of $\Theta(n)$.

Recall that $n = \mathcal{O}\left(\frac{V(\mathcal{S})}{V(tr)}\right)$ i.e. n is linear to the ratio ρ of the volumes of the space of motions and the transmission range of each mobile host. Thus,

Corollary 2 By forcing the support's head to move on a regular spanning subgraph of the motion graph, our protocol guarantees a total expected communication time of $\Theta(\rho)$, where ρ is the relative motion space size, and this time is independent of the total number of mobile hosts, and their movement.

Note 2 Our analysis assumed that the head of Σ moves according to a continuous time random walk of total rate 1 (rate of exit out of a node of G). If we select the support's hosts to be ψ *times faster* than the rest of the hosts, all the estimated times, except of the inter-support time, will be divided by ψ. Thus

Corollary 3 Our modified protocol where the support is ψ times faster than the rest of the mobile hosts guarantees an expected total communication time which can be made to be as small as $\Theta\left(\gamma \frac{\rho}{\sqrt{\psi}}\right)$ where γ is an absolute constant.

3.7.5 Time Efficiency - A Lower Bound

Lemma 6

$$m^* \;\geq\; \max_{i,j} E_i T_j$$

Proof: Consider the case where MH_S (or MH_R) just stands still on some vertex j and Σ's head starts at i. $\diamond$

Corollary 4 When Σ starts at positions according to the stationary distribution of its head's walk then

$$m^* \;\geq\; \max_{j} E_\pi T_j$$

for $j \in V$, where π is the stationary distribution.

From a Lemma of ([5], ch. 4, pp. 21), we know that for all i

$$E_\pi T_i \;\geq\; \frac{(1 \;-\; \pi(i))^2}{q_i \pi(i)}$$

where $q_i = d_i$ is the degree of i in G i.e.,

$$E_\pi T_i \;\geq\; \min_i \frac{(1 \;-\; \frac{d_i}{2m})^2}{d_i \frac{d_i}{2m}} \geq\; \min_i \frac{1}{2m} \frac{(2m \;-\; d_i)^2}{d_i^2}$$

For regular spanning subgraphs of G of degree Δ we have $m = \frac{\Delta n}{2}$, where $d_i = \Delta$ for all i. Thus,

Theorem 4 When Σ's head moves on a regular spanning subgraph of G, of m edges, we have that the expected meeting time of MH_S (or MH_R) and Σ cannot be less than $\frac{(n \;-\; 1)^2}{2m}$.

Corollary 5 Since $m = \Theta(n)$ we get a $\Theta(n)$ *lower bound* for the expected communication time. In that sense, our protocol's expected communication time is *optimal*, for a support size which is $\Theta(n)$.

3.7.6 Time Efficiency on the Average

Time-efficiency of semi-compulsory protocols for ad-hoc networks is not possible to estimate without a scenario for the motion of the mobile users not in the support (i.e. the non-compulsory part). In a way similar to [16, 31], we propose an "on-the-average" analysis by assuming that the movement of each mobile user *is a random walk on the corresponding motion graph* G. We propose this kind of analysis as a necessary and interesting first step in the analysis of efficiency of any semi-compulsory or even non-compulsory protocol for ad-hoc mobile networks. In fact, the assumption that the mobile users are moving randomly (according to uniformly distributed changes in their directions and velocities, or according to the random waypoint mobility model, by picking random destinations) has been used in [32], [29].

In the light of the above, we assume that any host not belonging in the support, conducts a random walk on the motion graph, independently of the other hosts.

The communication time of M_{Σ}^{S} is the total time needed for a message to arrive from a sending node u to a receiving node v. Since the communication time, T_{total}, between MH_S, MH_R is bounded above by $X+Y+Z$, where X is the time for MH_S to meet Σ, Y is the time for MH_R to meet Σ (after X) and Z is the message propagation time within Σ, in order to estimate the expected values of X and Y (these are random variables), we work as follows:

(a) Note first that X, Y are, statistically, of the same distribution, under the assumption that u, v are randomly located (at the start) in G. Thus $E(X) = E(Y)$.

(b) We now replace the meeting time of u and Σ by a hitting time, using the following thought experiment:
(b1) We fix the support Σ in an "average" place inside G.
(b2) We then collapse Σ to a single node (by collapsing its nodes to one but keeping the incident edges). Let H be the resulting graph, σ the resulting node and $d(\sigma)$ its degree.
(b3) We then estimate the hitting time of u to σ assuming u is somewhere in G, according to the *stationary distribution*, π of its walk on H. We denote the expected value of this hitting time by $E_\pi T_\sigma^H$.

Thus, now $E(T_{total}) = 2E_\pi T_\sigma^H + \mathcal{O}(k)$.

Note 3 The equation above amortizes over meeting times of senders (or receivers) because it uses the stationary distribution of their walks for their position when they decide to send a message.

Now, proceeding as in [5], we get the following lemma.

Lemma 7 For any node σ of any graph H in a continuous-time random walk

$$E_\pi T_\sigma^H \leq \frac{\tau_2(1-\pi(\sigma))}{\pi(\sigma)}$$

where $\pi(\sigma)$ is the (stationary) probability of the walk at node (state) σ and τ_2 is the relaxation time of the walk.

Note 4 In the above bound, $\tau_2 = \frac{1}{\lambda_2}$ where λ_2 is the second eigenvalue of the (symmetric) matrix $S = \{s_{i,j}\}$ where $s_{i,j} = \sqrt{\pi}(i)\ p_{i,j}\ (\sqrt{\pi(i)})^{-1}$ and $P = \{p_{i,j}\}$ is the transition matrix of the walk. Since S is symmetric, it is diagonalizable and the spectral theorem gives the following representation: $S = U\Lambda U^{-1}$ where U is orthonormal and Λ is a diagonal real matrix of diagonal entries $0 = \lambda_1 < \lambda_2 \leq \cdots \leq \lambda_{n'}$, $n' = n-k+1$ (where $n' = |V_H|$). These λ's are the eigenvalues of both P and S. In fact, λ_2 is an indication of the *expansion* of H and of the asymptotic rate of convergence to the stationary distribution, while relaxation time τ_2 is the corresponding interpretation measuring time.

It is a well-known fact (see e.g. [41]) that $\forall v \in V_H$, $\pi(v) = \frac{d(v)}{2m'}$ where $m' = |E_H|$ is the number of the edges of H and $d(v)$ is the degree of v in H. Thus $\pi(\sigma) = \frac{d(\sigma)}{2m'}$.

By estimating d(σ) and m' and remarking that the operation of locally collapsing a graph does not reduce its expansion capability and hence $\lambda_2^H \geq \lambda_2^G$ (see *Lemma 1.15* [21], ch. 1, p.13).

Theorem 5

$$E(X) = E(Y) \leq \frac{1}{\lambda_2(G)} \Theta\left(\frac{n}{k}\right)$$

Theorem 6 The expected communication time of our scheme is bounded above by the formula

$$E(T_{total}) \leq \frac{2}{\lambda_2(G)} \Theta\left(\frac{n}{k}\right) + \Theta(k)$$

Note 5 The above upper bound is minimized when k $= \sqrt{\frac{2n}{\lambda_2(G)}}$, a fact also verified by our experiments (see [16] and also section 3.8 in this Chapter).

We remark that this upper bound is tight in the sense that by [5], Chapter 3,

$$E_\pi T_\sigma^H \geq \frac{(1-\pi(\sigma))^2}{q_\sigma \pi(\sigma)}$$

where $q_\sigma = \sum_{j \neq \sigma} p_{\sigma j}$ in H_j is the total exit rate from σ in H. By using martingale type arguments we can show sharp concentration around the mean degree of σ.

3.7.7 Robustness

Now, we examine the robustness of the support management subprotocol M_Σ^S and M_Σ^R under single stop-faults.

Theorem 7 The support management subprotocol M_Σ^S is 1-fault tolerant.

Proof: If a single host of Σ fails, then the next host in the support becomes the head of the rest of the snake. We thus have two, independent, random walks in G (of the two snakes) which, however, will meet in expected time at most m^* (as in Theorem 2) and re-organize as a single snake via a very simple re-organization protocol which is the following:

When the head of the second snake Σ_2 meets a host h of the first snake Σ_1 then the head of Σ_2 follows the host which is "in front" of h in Σ_1, and all the part of Σ_1 after and including h waits, to follow Σ_2's tail. $\diamond$

Note that in the case that more than one faults occur, the procedure for merging snakes described above may lead to deadlock (see Figure 1).

Theorem 8 The support management subprotocol M_Σ^R is resilient to t faults, for any $0 \leq t < k$.

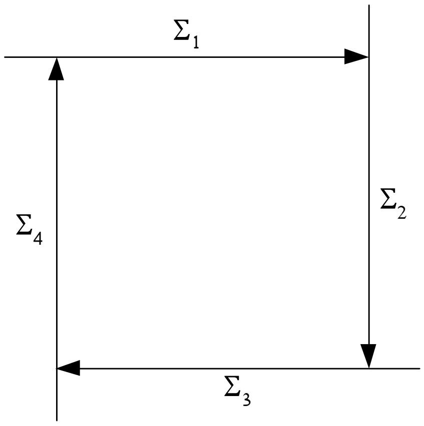

Figure 1: Deadlock situation when four "snakes" are about to be merged.

Proof: This is achieved using redundancy: whenever two runners meet, they create copies of all messages in transit. In the worst-case, there are at most $k - t$ copies of each message. Note, however, that messages may have to be re-transmitted in the case that only one copy of them exists when some fault occurs. To overcome this limitation, the sender will continue to transmit a message for which delivery has not been confirmed, to any other members of the support encountered. This guarantees that more than one copy of the message will be present within the support structure. $\diamond$

3.8 Experimental Evaluation

To experimentally validate, fine-tune and further investigate the proposed protocols, we performed a series of algorithmic engineering experiments.

All of our implementations follow closely the support motion subprotocols described above. They have been implemented as C++ classes using several advanced data types of LEDA [40]. Each class is installed in an environment that allows to read graphs from files, and to perform a network simulation for a given number of rounds, a fixed number of mobile users and certain communication and mobility behaviors. After the execution of the simulation, the environment stores the results again on files so that the measurements can be represented in a graphical way.

3.8.1 Experimental Set-up

A number of experiments were carried out modeling the different possible situations regarding the geographical area covered by an ad-hoc mobile network. We considered three kinds of inputs, unstructured (random) and more structured ones. Each kind of input corresponds to a different type of motion graph.

For all the cases that we investigated, each $h \notin \Sigma$, generates a new message with probability $p_h = 0.01$ (i.e., a new message is generated roughly every 100 rounds) by picking a random destination host. The selection of this value for p_h is based on the assumption that the mobile users do not execute a real-time application that requires continuous exchange of messages. Based on this choice of p_h, each experiment takes several thousands of rounds in order to get an acceptable average message delay. For each experiment, a total of $100,000$ messages were transmitted. We carried out each experiment until the 100,000 messages were delivered to the designated receivers. Our test inputs are as follows.

Random Graphs. This class of graphs is a natural starting point in order to experiment on areas with obstacles. We used the $G_{n,p}$ model obtained by sampling the edges of a complete graph of n nodes independently with probability p. We used $p = \frac{1.05 \log n}{n}$ which is marginally above the connectivity threshold for $G_{n,p}$. The test cases included random graphs with $n \in \{1600, 3200, 6400, 40000, 320000\}$ over different values of $k \in [3, 45]$.

2D Grid Graphs. This class of graphs is the simplest model of motion one can consider (e.g., mobile hosts that move on a plane surface). We used two different $\sqrt{n} \times \sqrt{n}$ grid graphs with $n \in \{400, 1600\}$ over different values of k. For example, if $\mathcal{V}(tc) = 50m^2$ then the area covered by the ad-hoc network will be $20,000m^2$ in the case where $n = 400$.

Bipartite multi-stage graphs. A bipartite multi-stage graph is a graph consisting of a number ξ of stages (or levels). Each stage contains n/ξ vertices and there are edges between vertices of consecutive stages. These edges are chosen randomly with some probability p among all possible edges between the two stages. This type of graphs is interesting as such graphs may model movements of hosts that have to pass through certain places or regions, and have a different second eigenvalue than grid and $G_{n,p}$ graphs (their second eigenvalue lies between that of grid and $G_{n,p}$ graphs). In our experiments, we considered $\xi = \log n$ stages and chose $p = \frac{n}{\xi} \log \frac{n}{\xi}$ which is the threshold value for bipartite connectivity (i.e., connectivity between each pair of stages). The test cases included multi-stage graphs with 7, 8, 9 and 10 stages, number of vertices $n \in$\{1600, 3200, 6400, 40000\}, and different values of $k \in [3, 45]$.

3.8.2 Validation of the SNAKE Protocol's Performance

In [12, 16] we performed extensive experiments to evaluate the performance of the "snake" protocol. A number of experiments were carried out modeling the different possible situations regarding the geographical area covered by an ad-hoc mobile network. We only considered the *average case where all users (even those not in the support) perform independent and concurrent random walks.*

In the first set of experiments, we focused on the validation of the theoretical bounds provided for the performance of the "Snake" protocol. For each message generated, we calculated the delay X (in terms of rounds) until the message has been transmitted to a member of Σ. Additionally, for each message received by the support, we logged the delay Y (in terms of rounds) until the message was transmitted to the receiver. We assumed that no extra delay was imposed by the mobile support hosts (i.e. $T_\Sigma = 0$). We then used the average delay over all measured fractions of message delays for each experiment in order to compare the results and further discuss them.

The experimental results have been used to compare the average message delays ($E(T_{total})$) with the upper bound provided by the theoretical analysis of the protocol. We observe that as the total number n of motion-graph nodes remains constant, as we increase the size k of Σ, the total message delay (i.e. $\mathrm{E}(T_{total})$) is decreased. Actually, $\mathrm{E}(T_{total})$ initially decreases very fast with k, while having a limiting behavior of no further significant improvement when k crosses a certain threshold value (see e.g. figure 2). Therefore, taking into account a possible amount of statistical error, the following has been experimentally validated:

$$\text{if } k_1 \; > \; k_2 \; \Rightarrow \; E_1(T_{total}) \; < \; E_2(T_{total})$$

Throughout the experiments we used small and medium sized graphs, regarding the total number of nodes in the motion graph. Over the same type of graph (i.e. grid graphs) we observed that the average message delay $\mathrm{E}(T_{total})$ increases as the total number of nodes increases. This

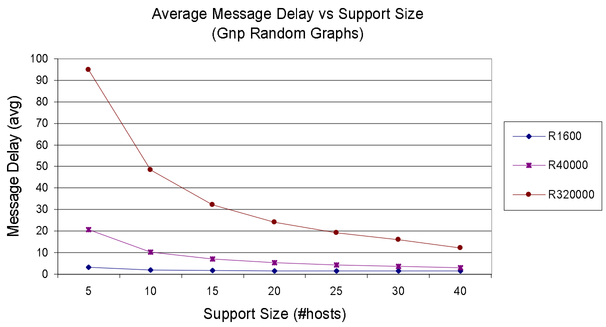

Figure 2: Average message delay for the "Snake" approach over $G_{n,p}$ types of motion graphs with different sizes n and varied support size k.

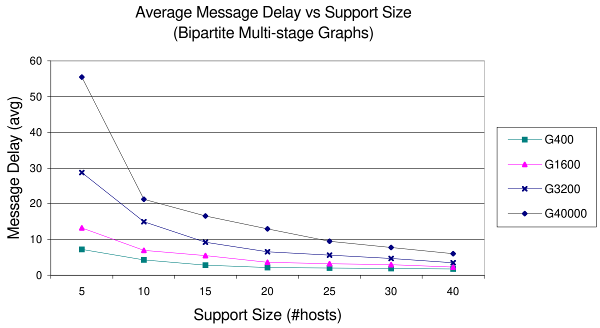

Figure 3: Average message delay of the "Snake" approach over bipartite multi-stage types of motion graphs with different sizes (n) and varied support size k.

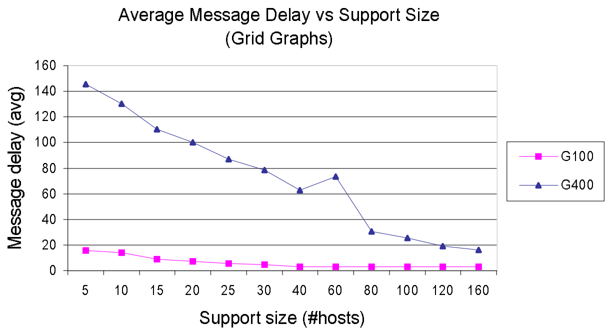

Figure 4: Average message delay of "Snake" approach over grid types of motion graphs with different sizes (n) and varied support size k.

can be clearly seen from figure 3 where the curves of the four different graph-sizes are displayed. It is safe to assume that for the same graph type and for a fixed-size Σ the overall message delay increases with the size of the motion graph. We further observe that $\mathrm{E}(T_{total})$ does not increase linearly but is affected by the expansion rate of the graph, or otherwise, as stated in the theoretical analysis, by the second eigenvalue of the matrix of the graph. If we take into account possible amount of statistical error that our experiments are inherently prone to, we can clearly conclude the following:

$$\text{if } n_1 \; > \; n_2 \; \Rightarrow \; E_1(T_{total}) \; > \; E_2(T_{total})$$

Furthermore, we remark that $\mathrm{E}(T_{total})$ does not only depend on the actual size of the graph and the size of the support Σ but, additionally, that it is directly affected by the type of the graph. This is expressed throughout the theoretical analysis by the effect of the second eigenvalue of the matrix of the graph on communication times. If we combine the experimental results of figures 2, 3 and 4 for fixed number of motion-graph nodes (e.g. $n = 400$) and fixed number of Σ size (e.g. $k = 40$) we observe the following:

$$E_{grid}(T_{total}) \; > \; E_{multistage}(T_{total}) \; > \; E_{G_{n,p}}(T_{total})$$

which validates the corresponding results due to the analysis.

In section 3.7.6, we note that the upper bound on communication times $E(T_{total}) \leq \frac{2}{\lambda_2(G)} \Theta\left(\frac{n}{k}\right) + \Theta(k)$ is minimized when $k = \sqrt{\frac{2n}{\lambda_2(G)}}$. Although our experiments disregard the delay imposed by Σ (i.e. we take $\Theta(k) \to 0$) this does not significantly affect our measurements. Interestingly, the values of figures 2, 3 and 4 imply that the analytical upper bound is not very tight. In particular, for the $G_{n,p}$ graphs we clearly observe that the average message delay drops below 3 rounds (which is a very low, almost negligible, delay) if Σ size is equal to 10; however, the above formula implies that this size should be higher, especially for the medium sized graphs. Actually, it is further observed that there exists a certain threshold value (as also implied by the theoretical analysis) for the size of Σ above which the overall message delay does not further significantly improve. The threshold size for the support validated experimentally is smaller than the threshold implied by the analytical bound.

3.8.3 Experimental Comparison of the SNAKE and the RUNNERS Protocols

In [17] we performed a comparative experimental study of the "snake" and the "runners" protocols. Our test inputs were similar to that used in section 3.8.2. We measured the total delay of

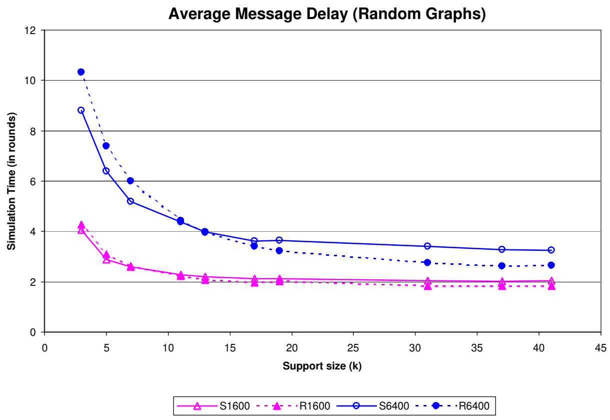

Figure 5: Average message delay over random graphs with different sizes (n) and varied support size k, for the Snake (S) protocol and Runners (R) protocol.

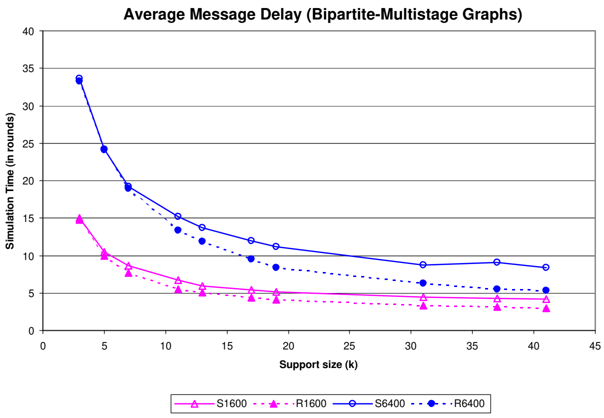

Figure 6: Average message delay over bipartite multi-stage motion graphs with different sizes (n) and varied support size k, for the Snake (S) protocol and Runners (R) protocol.

messages exchanged between pairs of sender-receiver users. For each motion graph constructed, we injected $1,000$ users (mobile hosts) at random positions that generated 100 transaction message exchanges of 1 packet each by randomly picking different destinations. For each message generated, we calculated the overall delay (in terms of simulation rounds) until the message was finally transmitted to the receiver. We used these measurements to experimentally evaluate the performance of the two different support management subprotocols.

The reported experiments for the three different test inputs we considered are illustrated in Figures 5–7. In these figures, we have included the results of two instances of the input graph w.r.t. n, namely a smaller and a larger value of n (similar results hold for other values of n). Each curve in the figures is characterized by the name 'Px', where P refers to the protocol used (S for snake and R for runners) and x is a 3 or 4 digit number denoting the value of n.

The curves reported for the snake protocol confirm the theoretical analysis in section 3.7.6. That is, the average message delay drops rather quickly when k is small, but after some threshold value stabilizes and becomes almost independent of the value of k. This observation applies to all test inputs we considered. We also note that this behavior of the snake protocol is similar to the one reported in 3.8.2, although we count here the extra delay time imposed by the

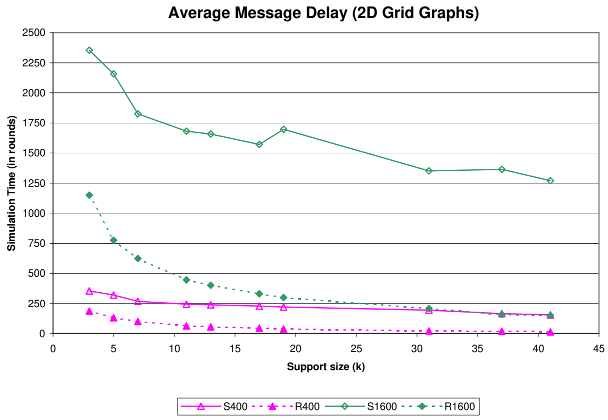

Figure 7: Average message delay over grid motion graphs with different sizes (n) and varied support size k, for the Snake (S) protocol and Runners (R) protocol.

synchronization subprotocol P_3.

Regarding the "runners" protocol, we firstly observe that its curve follows the same pattern with that of the snake protocol. Unfortunately, we don't have yet a theoretical analysis for the new protocol to see whether it behaves as should expected, but from the experimental evidence we suspect that it obeys a similar but tighter bound than that of the snake protocol. Our second observation is that the performance of the "runners" protocol is slightly better than that of the "snake" protocol in random graphs (except for the case of random graphs with small support size; see figure 5) and bipartite multi-stage graphs (figure 6), but is substantially better in the more structured case (grid graphs; see figure 7). This could be partly explained by the fact that the structured graphs have a smaller second eigenvalue than that of bipartite multi-stage and random graphs. A small value for this eigenvalue makes the average message delay to be more dependent on the hidden constant in the asymptotic analysis of the "snake" protocol (see section 3.7.6) which is apparently larger than the corresponding constant of the runners protocol.

4 An Algorithm for Reconfiguration of Robot Chains

4.1 Introduction

The aspect of motion planning in robotic systems has become recently a topic of significant research and development activity. We study here a particular motion planning problem, that of reconfiguration. Roughly speaking, the reconfiguration problem for a robotic system comprised of several modules (robots) consists in reshaping the system by motions of the robots that change their relative positions and thus allow the system to go from an initial configuration to a desired goal configuration.

A reason for the increasing interest in the reconfiguration problem of robotic systems is its wide range of important applications. To name a few characteristic examples, reshaping of robotic systems may allow such systems to perform specific shape-dependent tasks (e.g. "bridge-like" shapes may be useful in crossing rivers, adjustable buttress shapes may be used in supporting collapsing buildings etc.). Also, the operation of systems of robots in areas where human presence is either impossible or dangerous (such as in disaster places, or deep undersea areas and space missions) necessitates the ability of such systems to self-organize themselves into (possibly changing) formations and shapes. Furthermore, the reshaping feature significantly increases

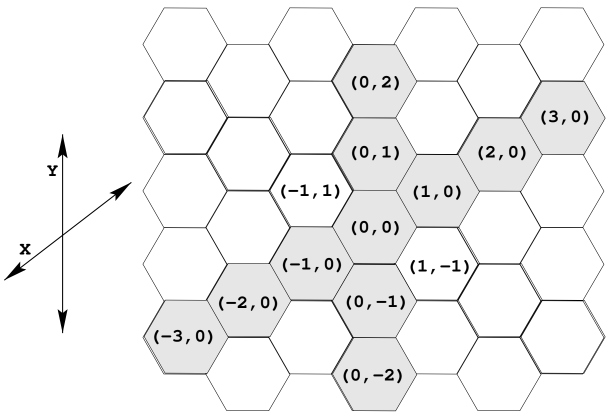

Figure 8: The Coordinate System

the fault-tolerance of robotic systems since it allows them to better adapt to external (possibly adversarial) conditions or to dynamically co-ordinate responsibilities in case of failure of certain robots in the system.

Such robotic systems, able to change their shape by individual robot motions, are called self-configurable or metamorphic. More specifically, a self-configurable robotic system is a collection of independently controlled, mobile modules (robots), each of which can connect, disconnect and move around adjacent modules. A metamorphic robotic system is a special case of self-configurable systems, where additionally all modules (robots) are identical (in structure, motion and computing capabilities).

For achieving shape formation, the modules should be able to pack together as densely as possible (i.e. by minimizing gaps between modules). A module shape allowing their dense packing with each other, is one having both regularity and symmetry. A typical example of such module structure (studied here as well) is that of regular hexagons of equal size.

Two major, different approaches in the design and operation of metamorphic systems use a) centralized motion planning algorithms b) distributed algorithms. Most of the existing strategies are centralized. The fundamental work of [47] that we present in this section is in fact the first distributed approach to reconfiguration in metamorphic robotic systems. We wish to note the following three basic advantages of a distributed approach over a centralized one: a) the motion decisions are local, thus such algorithms can operate under lack of knowledge of the global system structure and are potentially more efficient b) the algorithm's fault-tolerance is greater compared to the centralized case, where failure of the central module is fatal c) distributed algorithms can operate under weaker modelling assumptions, such as that of lack of communication between modules.

4.2 The Model

The modelling assumptions in [47] include basically the coordination system as well as the hardware and motion characteristics of the robots.

4.2.1 The Coordinate System

The two-dimensional (plane) area is partitioned into regular hexagonal cells of equal size. The coordinate system exemplified in figure 8 is used for labelling the cells.

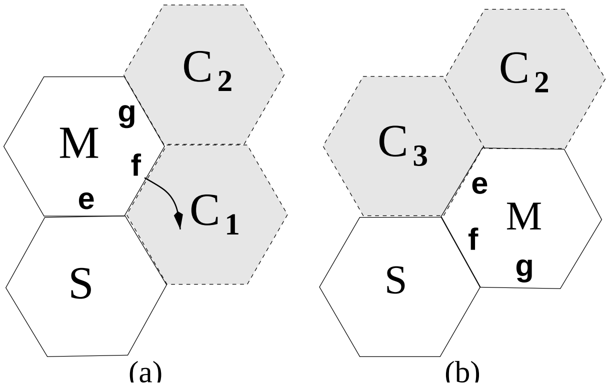

Figure 9: Example of Movement

A key concept for motion planning by the reconfiguration algorithm, is that of the "lattice distance" in the cell structure. Given the coordinates of two cells, $c_1 = (x_1, y_1)$ and $c_2 = (x_2, y_2)$, the lattice distance, LD, between them is defined as follows: Let $\Delta x = x_1 - x_2$ and $\Delta y = y_1 - y_2$. If $\Delta x \cdot \Delta y < 0$, then $LD(c_1, c_2) = \max\{|\Delta x|, |\Delta y|\}$. Otherwise, $LD(c_1, c_2) = |\Delta x| + |\Delta y|$.

The lattice distance of cells in the plane area is very important, since it is equal to the minimum number of cells a robot must move through to go from the one cell to the other. As an example, consider cells $c_1 = (0, 2)$ and $c_2 = (-1, 1)$. It is $\Delta x = 1, \Delta y = 1$, so $LD(c_1, c_2) = |\Delta x| + |\Delta y| = 2$. Consider now c_1 and a third cell $c_3 = (1, -1)$. If is $\Delta x = -1, \Delta y = 3$, so $LD(c_1, c_3) = \max\{|\Delta x|, |\Delta y|\} = 3$.

4.2.2 The Modules (Robots)

The model in [47] includes the following assumptions about the modules (imposed by hardware constraints and application needs):

- Modules are identical in termes of computing capability.
- Each module runs the same program.
- Each module is a regular hexagon of the same size as the cells of the plane area. At any point in time each module occupies exactly one of the cells.
- Each module knows at all times the following: a) its location (i.e. the x, y coordinates of the cell that it currently resides on), b) its orientation (which edge it is facing in which direction), and c) which of its neighboring cells is occupied by another module (the latter piece of information is assumed to be provided by an appropriate link layer protocol).

Furthermore, [47] makes the following assumptions about the possible motion of the robots:

- Modules move in synchronous rounds.
- A module M is capable of moving to an adjacent cell, C_1, if and only if the following conditions hold: a) cell C_1 is empty, b) module M has a neighbor S that does not move in the current round (called the substrate) and S is also adjacent to cell C_1 and c) the neighboring cell to M on the other side of C_1 from S, C_2, is empty. See figure 9 for an example. Note that the algorithm must ensure that each module trying to move has

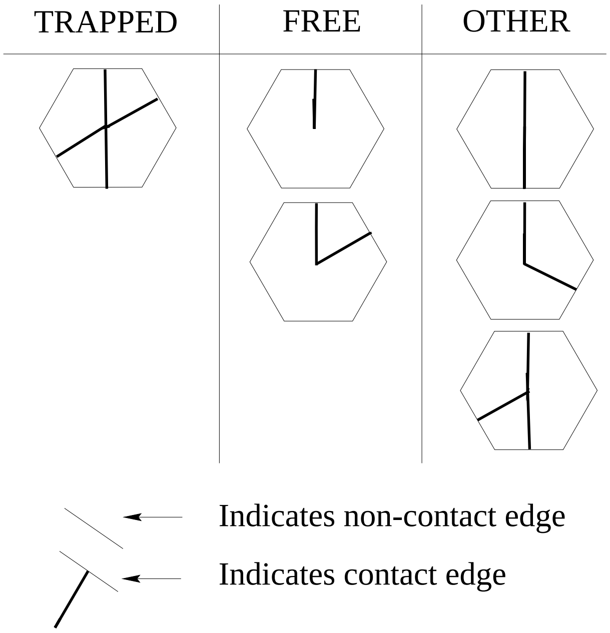

Figure 10: Module types

a substrate that is not moving, otherwise the algorithm may engage in unpredictable behaviour.

- Only one module tries to move into a particular cell in each round. Note that if the algorithm does not ensure collision freedom, then unpredictable behaviour may emerge.

4.2.3 The Problem

The problem consists in designing a distributed algorithm running locally at each module, putting the modules to move from an initial configuration, I, to a (known to all modules) goal configuration, G.

We here in fact deal with a special case of the problem where the initial and the goal configurations are straight-line sets of modules that overlap in exactly one cell. See figure 11 for an example of a reconfiguration.

4.2.4 The Algorithm

In this section we present the algorithm of Walter, Welch and Amato ([47]) only for the case where the initial configuration and the goal configuration are collinear. The interested reader can find in their paper an extension of this algorithm to the non-collinear case (i.e. when the goal configuration is allowed to intersect the initial one in any orientation).

[47] categorizes modules based on their possible types of edge contacts during an execution of the algorithm. In particular, modules are classified into three categories (trapped, free, and other) according to the number and orientation of their contact and non-contact edges.

Non-contact edges are those edges where the module is adjacent to an empty cell. Contact edges connect a module to an occupied adjacent cell. In the graphical presentation in figure 10, there are vertical line segments towards contact edges. We note that the classification above applies to any rotation of a module.

Trapped modules can not move because of the number and orientation of their contact edges (this represents hardware constraints on movement, explained in Section 4.2.2). Free modules are required to move in the algorithm. The other category includes modules whose movement is restricted by the algorithm, even though their movement is not constrained by the number and orientation of contact edges.

The algorithm in [47] operates under the following assumptions:

a) Each module knows the total number n of modules in the system. Modules also know the goal configuration G.

b) Initially, exactly one module occupies each cell of the initial configuration I.

c) Configrations I and G are collinear, while they overlap in exactly one cell.

The algorithm works in synchronous rounds. Each module moves (throughout the execution) either clockwise (CW) or anticlockwise (CCW). This direction of movement for each module is decided by the algorithm in a way that creates alternate direction for adjacent modules. This is achieved by using the parity of each module's distance from the (single) overlap cell.

In each round, each module calculates whether it is free, and if free it moves in the direction (CW or CCW) calculated initially.

We below provide the pseudo-code description (as given in [47]) of the reconfiguration algorithm. Before, we present some data structures kept at each module and used in the algorithm.

-The boolean array *contacts* shows which edges have neighboring modules. (It is assumed that this information is accordingly updated at each round by some underlying low layer protocol).

- The coordinates of a module according to the coordinate system explained earlier are kept in data structure *coord*.

- The array *goalCell* contains all coordinates of cells in the goal configuration in decreasing order of the y coordinate.

- Variable d keeps the direction of movement, CW or CCW.

- *flips* is a counter used to represent whether a module is free or not.

The pseudo-code of the reconfiguration algorithm follows:

Each module $\notin G$ executes:

Initially:

if $((n - LD(myCoord, goalCell[1]))$ is even)

then $d = CCW$

else $d = CW$

In round $r = 1, 2, \ldots$:

if $(isFree())$

then move d

Procedure $isFree()$:

$flips = 0$

for $(i = 0$ to $5)$ do

if $(contacts[i] \neq contacts[(i+1)\%6])$

then $flips++$

return ($flips == 2$)

Note that modules in goal cells are not allowed to move. In this sense the above algorithm

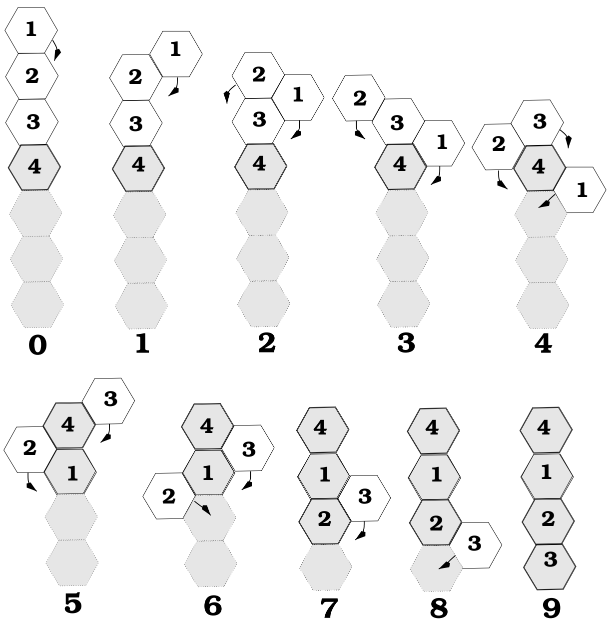

Figure 11: Reconfiguration Example

is called a "goal-stopping" algorithm.

Figure 11 presents an execution of the above reconfiguration algorithm when the number of robots in the system is $n = 4$. In this figure, occupied cells have solid borders and goal cells are shaded. For purposes of analysis, modules are labeled 1 through 4. Remark that nine rounds are required for this reconfiguration.

4.2.5 Analysis

We here present sketches of the correctness and performance proofs of [47].

Assume, without loss of generality, that configurations I and G are directed from north to south and also that I is north of G. For reference clarity in the following proofs, number the cells in I and G from 1 through $2n - 1$ from north to south. We will refer to the module occupying at the beginning of the algorithm cell i as module i, where $1 \leq i \leq n$. We will refer to a cell's neighboring cells as north (N), northeast (NE), southeast (SE), south (S), southwest (SW), and northwest (NW).

Theorem 9 [47] The reconfiguration algorithm is correct.

Proof (sketch following [47]: We show that the following properties $P1 - P3$ are invariant throughout the execution. For each $i, 1 \leq i \leq n$, and each round $S \geq 1$, at the end of round S, it is:

- P1: if $S < 2i - 1$, then module i is in cell i.
- P2: if $S = 2i - 1 + j$, $0 \leq j \leq n - 1$, then
  a) if i is odd, then module i is SE of cell $i + j$,
  b) if i is even, then module i is SW of cell $i + j$,
- P3: if $S \geq 2i - 1 + n$, then module i is in cell $i + n$.

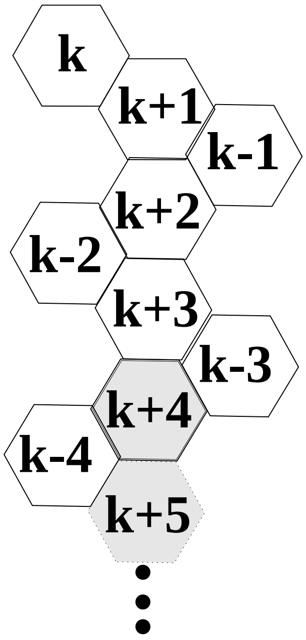

Figure 12: Configuration at end of round $2k-1$, where k is even

The proof is by induction on the number of rounds, S, in the execution.

The basis ($S = 0$) is easy since initially $P2$ and $P3$ are not applicable, and $P1$ is clearly true by assumption.

For the inductive hypothesis, assume that the invariants hold for round $S-1, S > 0$. Figure 12 shows the configuration at the end of round $2k-1$. In this figure, occupied cells have solid borders and goal cells are shaded.

Without loss of generality, we take i to be even.

Case 1: $S < 2i - 1$.

Thus, $S-1 < 2i-2$. By the inductive hypothesis and $P1$, module $i-1$ occupies cell $i-1$ and module $i+1$ is in cell $i+1$ at the end of round $S-1$. Therefore, module i does not move in round S because it will not be free in this round. Referring to Figure 12, module $k+2$ has contacts that correspond to those described for module i. By property $P1$, module i is in cell i at the end of round $S-1$, thus (since it does not move in next round) it remains in cell i at the end of round S.

Case 2: $S = 2i - 1 + j$ for some j, $0 \leq j \leq n-1$.

Thus, $S - 1 = 2i - 1 + (j-1) = 2(i-1) + j + 1$.

If $j = 0$, then by the inductive hypothesis $P1$ implies that module i is in cell i at the end of round $S-1$ and because of $P2$ module $i-1$ is SE of cell i at the end of round $S-1$, since $S-1 = 2(i-1)+0+1$. Thus module i is free at the end of round $S-1$. So module i moves anticlockwise, and gets SW of cell i in round S. In Figure 12, module k corresponds to the position described for module i and module $k-1$ corresponds to the position described for module $i-1$ in round S.

If $j > 0$, then property $P1$ implies that module i is in the cell SW of cell $i+(j-1)$ at the end of round $S-1$. Module i will be free at the end of round $S-1$ because i will have contacts only on its NE and SE edges. Therefore, module i will move CCW in round S to be SW of cell

j. In Figure 12, module $k-2$ is like module i at the end of round $S-1$.

Case 3: $S \geq 2i - 1 + n$.

Then for the equality case it is $S-1 = 2i-2+n$, thus by $P3$ module $i-1$ is in cell $i+n-1$ in round $S-1$, so module $i-1$ will not move in round $S-1$ or later. By $P2$, module i is in the cell SW of cell $i+n-1$ at the end of round $S-1$. Therefore, module i is free at the end of round $S-1$ because it has only one contact. Module i has no other neighbors in round $S-1$. So in round S, module i moves to cell $i+n$ and stops moving in this round by the code. In Figure 12, module $k-4$ is in a position like that described for module i, and module $k+4$ is like that $i-1$ at the end of round $S-1$.

If $S > 2i - 1 + n$, then, since at the end of round S module i is already in cell $i+n$ by $P3$, then it is still in cell $i+n$ in every round following S. By the code, once a module is in a goal position, it does not move further. $\diamond$

The above invariants imply that the modules only use three columns of the cell structure. At the initial configuration, modules are all in the center column and, during the execution, modules in outer columns are spaced or staggered such that there is an empty cell between any two modules. Invariant $P3$ implies that, after round $3(n-1)$, all modules are in goal positions.

As far as the performance of the algorithms is concerned, [47] proves the following:

Theorem 10 ([47]) *The reconfiguration algorithm takes $3(n-1)$ rounds and makes n^2-1 module movements.*

Proof: Clearly, invariants $P1$ through $P3$, imply that the algorithm takes $3(n-1)$ rounds. Invariants $P2$ and $P3$ show that $n-1$ (all but the overlapping one) of the modules in the initial configuration make $n+1$ moves each, thus we get a total of n^2-1 module movements.

$\diamond$

By estimating lower bounds, the authors in [47] prove that their algorithm is asymptotically optimal, both with respect to number of moves and rounds (time).

References

[1] M. Adler, H. Racke, C. Sohler, N. Sivadasan, and B. Vocking: Randomized Pursuit-Evasion in Graphs. In *Proc. 29th ICALP, 901-912*, 2002.

[2] M. Adler and C. Scheideler: Efficient Communication Strategies for Ad-Hoc Wireless Networks. In *Proc. 10th Annual Symposium on Parallel Algorithms and Architectures* – SPAA'98, 1998.

[3] M. Ajtai, J. Komlos and E. Szemeredi: Deterministic Simulation in LOGSPACE. In *Proc. 19th Annual ACM Symposium on Theory of Computing* – STOC'87, 1987.

[4] D. Aldous and P. Diaconis: Strong Stationary Times and Finite Random Walks. Adv. Appl. Math., 8:69-97, 1987.

[5] D. Aldous and J. Fill: *Reversible Markov Chains and Random Walks on Graphs.* Unpublished manuscript. http://stat-www.berkeley.edu/users/aldous/book.html, 1999.

[6] N. Alon and F.R.K. Chung: Explicit Construction of linear sized tolerant networks. Discrete Mathematics, vol 72, pp 15-19, 1998.

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

[8] G. Brightwell and P. Winkler: Maximum Hitting Times for Random Walks on Graphs. Journal of Random Structures and Algorithms, 1:263-276, 1990.

[9] J. Broch, D.B. Johnson, and D.A. Maltz: The dynamic source routing protocol for mobile ad-hoc networks. IETF, Internet Draft, draft-ietf-manet-dsr-01.txt, December, 1998.

[10] A. Boukerche: A Source Route Discovery Optimization Scheme in Mobile Ad hoc Networks. In *Proc. 8th International Conference on Parallel and Distributed Computing* – EUROPAR'02, 2002.

[11] C. Busch, S. Surapaneni, and S. Tirthapura: Analysis of Link Reversal Routing Algorithms for Mobile Ad Hoc Networks. In *Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2003)*, pp. 210-219.

[12] I. Chatzigiannakis, S. Nikoletseas and P. Spirakis: Distributed Communication and Control Algorithms for Adhoc Mobile Networks. In the Journal of Parallel and Distributed Computing (JPDC), Special Issue on Mobile Ad-hoc Networking and Computing, (63) (2003) pp. 58–74, 2003.

[13] I. Chatzigiannakis and S. Nikoletseas: Design and Analysis of an Efficient Communication Strategy for Hierarchical and Highly Changing Ad-hoc Mobile Networks. In ACM/Baltzer Mobile Networks Journal (MONET), Special Issue on Parallel Processing Issues in Mobile Computing, Guest Editors: A. Zomaya and M. Kumar, to appear in 2003.

[14] I. Chatzigiannakis, S. Nikoletseas, and P. Spirakis: An Efficient Communication Strategy for Ad-hoc Mobile Networks. In *Proc. 15th International Symposium on Distributed Computing* – DISC'01. Lecture Notes in Computer Science, Volume 2180 (Springer-Verlag, 2001), pp. 285–299. Also Brief Announcement in *Proc. 20th Annual Symposium on Principles of Distributed Computing* – PODC'01, pp. 320–322, ACM Press 2001.

[15] I. Chatzigiannakis, S. Nikoletseas, and P. Spirakis: On the Average and Worst-case Efficiency of Some New Distributed Communication and Control Algorithms for Ad-hoc Mobile Networks. Invited Paper in the Proc. 1st ACM International Workshop on Principles of Mobile Computing, pp. 1–16, (POMC'01)(2001).

[16] I. Chatzigiannakis, S. Nikoletseas, and P. Spirakis: Analysis and Experimental Evaluation of an Innovative and Efficient Routing Approach for Ad-hoc Mobile Networks. In *Proc. 4th Annual Workshop on Algorithmic Engineering* – WAE'00, 2000. Lecture Notes in Computer Science, Volume 1982 (Springer-Verlag, 2000).

[17] I. Chatzigiannakis, S. Nikoletseas, and P. Spirakis: Experimental Evaluation of Basic Communication for Ad-hoc Mobile Networks. In *Proc. 5th Annual Workshop on Algorithmic Engineering* – WAE'01, pp. 159–171, 2001. Lecture Notes in Computer Science, Volume 2141 (Springer-Verlag, 2001).

[18] I. Chatzigiannakis, S. Nikoletseas, and P. Spirakis: An Efficient Routing Protocol for Hierarchical Ad-Hoc Mobile Networks In *Proc. 1st International Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing*, satellite workshop of *15th Annual International Parallel & Distributed Processing Symposium* – IPDPS'01, 2001.

[19] I. Chatzigiannakis and S. Nikoletseas: An Adaptive Compulsory Protocol for Basic Communication in Highly Changing Ad-hoc Mobile Networks. In *Proc. 2nd International Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing*, Satellite Workshop of *16th Annual International Parallel & Distributed Processing Symposium* – IPDPS'02.

[20] G. Chirikjian: Kinematics of a metamorphic robotic system. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 449 – 455, 1994.

[21] F.R.K. Chung: Spectral Graph Theory. Regional conference series in mathematics, ISSN 0160-7642; no. 92. CBMS Conference on Recent Advances in Spectral Graph Theory, California State University at Fesno, 1994. ISBN 0-8218-0315-8

[22] D. Coppersmith, P. Tetali and P. Winkler: Collisions among Random Walks on a Graph. SIAM J. Disc. Math., 6:363-374, 1993.

[23] D. Coppersmith, P. Doyle, P. Raghavan and M. Snir: Random Walks on Weighted Graphs and Applications to On-line Algorithms. In *Proc. 22nd ACM Symposium on Theory of Computing* – STOC'00, pp. 369–378, 1990.

[24] J. Diaz, J. Petit, M. Serna: A random graph model for optical networks of sensors. In *IEEE Transactions Mobile Computing, Vol 2, No. 3 186–196*, 2003

[25] G. Dommety and R. Jain: Potential networking applications of global positioning systems (GPS). Tech. Rep. TR-24, CS Dept., The Ohio State University, April, 1996.

[26] R. Durret: *Probability: Theory and Examples.* Wadsworth, 1991.

[27] P. Diaconis and L. Saloff-Coste: Comparison Theorems for reversible Markov Chains. The Annals of Applied Probability, Vol. 3, No. 3, pp. 696-730, 1993.

[28] E. Gafni and D. Bertsekas: Distributed algorithms for generating loop-free routes in networks with frequently changing topology. IEEE Transactions on Communications, C-29(1):11-18, 1981.

[29] Z.J. Haas and M.R. Pearlman: The performance of a new routing protocol for the reconfigurable wireless networks. In *Proc. of the IEEE International Conference on Communications* – ICC'98, 1998.

[30] Z.J. Haas and M.R. Pearlman: The zone routing protocol (ZRP) for ad-hoc networks. IETF, Internet Draft, draft-zone-routing-protocol-02.txt, June, 1999.

[31] K.P. Hatzis, G.P. Pentaris, P.G. Spirakis, V.T. Tampakas and R.B. Tan: Fundamental Control Algorithms in Mobile Networks. In *Proc. 11th Annual ACM Symposium on Parallel Algorithms and Architectures* – SPAA'99, 1999.

[32] G. Holland and N. Vaidya: Analysis of TCP Performance over Mobile Ad Hoc Networks. In *Proc. 5th Annual ACM/IEEE International Conference on Mobile Computing* – MOBICOM'99, 1999.

[33] T. Imielinski and H.F. Korth: *Mobile Computing.* Kluwer Academic Publishers, 1996.

[34] R. Iyer, D. Karger, H. Rahul, and M. Thorup. An experimental study of poly-logarithmic fully-dynamic connectivity algorithms. In *Proc. 2nd Workshop on Algorithm Engineering and Experiments* – ALENEX'00, pp. 59–78, 2000.

[35] Y. Ko and N.H. Vaidya: Location-Aided Routing (LAR) in Mobile Ad-hoc Networks. In *Proc. 4th Annual ACM/IEEE International Conference on Mobile Computing* – MOBICOM'98, 1998.

[36] Q. Li and D. Rus: Sending Messages to Mobile Users in Disconnected Ad-hoc Wireless Networks. In *Proc. 6th Annual ACM/IEEE International Conference on Mobile Computing* – MOBICOM'00, 2000.

[37] N. Malpani, N. Vaidya and J.L. Welch: Distributed Token Circulation on Mobile Ad Hoc Networks. In *Proc. 9th International Conference on Network Protocols (ICNP)*, 2001.

[38] N. Malpani, J.L. Welch and N. Vaidya: Leader Election Algorithms for Mobile Ad Hoc Networks. In *Proc. 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL M for Mobility),* pp. 96–103, 2000.

[39] N.A. Lynch: *Distributed Algorithms.* Morgan Kaufmann Publishers Inc., 1996.

[40] K. Mehlhorn and S. Näher: *LEDA: A Platform for Combinatorial and Geometric Computing.* Cambridge University Press, 1999.

[41] R. Motwani and P. Raghavan: *Randomized Algorithms.* Cambridge University Press, 1995.

[42] V.D. Park and M.S. Corson: Temporally-ordered routing algorithms (TORA) version 1 functional specification. IETF, Internet Draft, draft-ietf-manet-tora-spec-02.txt, October, 1999.

[43] C.E. Perkins: *Ad Hoc Networking.* Addison-Wesley, Boston, USA, January, 2001.

[44] C.E. Perkins and E.M. Royer: Ad-hoc on demand distance vector (AODV) routing. IETF, Internet Draft, draft-ietf-manet-aodv-04.txt, 1999.

[45] S. Ross: Stochastic Processes, John Wiley & Sons, Inc., 1996.

[46] E. Szemeredi: Regular Partitions of graphs. Colloques Internationaux C.N.R.S., Nr 260, Problemes Combinatoires et Theorie des Graphes, Orsay, pp. 399-401, 1976.

[47] J.E. Walter, J.L. Welch and N.M. Amato: Distributed Reconfiguration of Metamorphic Robot Chains In *Proc. 19th ACM Annual Symposium on Principles of Distributed Computing* – PODC'00, 2000.

[48] J. Walter, J. Welch and N. Vaidya: A Mutual Exclusion Algorithm for Ad Hoc Mobile Networks. accepted to Wireless Networks. Also in the Proc. of 2nd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL M for Mobility), 1998.

[49] Y. Zhang and W. Lee: Intrusion Detection in Wireless Ad-hoc Networks. In *Proc. 6th Annual ACM/IEEE International Conference on Mobile Computing* – MOBICOM'00, 2000.

[50] Iowa State University GPS page. http://www.cnde.iastate.edu/ gps.html.

[51] NAVSTAR GPS operations. http://tycho.usno.navy.mil/gpsinfo.html.