Alcom-FT Technical Report Series
ALCOMFT-TR-03-178

Algorithmic Design for Communication in Mobile Ad hoc Networks*

Azzedine Boukerche[1] and Sotiris Nikoletseas[2]

[1] School of Information Technology and Engineering (SITE)
University of Ottawa, Canada
Email: boukerch@site.uottawa.ca
[2] Department of Computer Engineering and Informatics
University of Patras, and Computer Technology Institute (CTI), Greece
Email: nikole@cti.gr

Abstract. As a result of recent significant technological advances, a new computing and communication environment, Mobile Ad Hoc Networks (MANET), is about to enter the mainstream. A multitude of critical aspects, including mobility, severe limitations and limited reliability, create a new set of crucial issues and trade-offs that must be carefully taken into account in the design of robust and efficient algorithms for these environments. The communication among mobile hosts is one among the many issues that need to be resolved efficiently before MANET becomes a commodity.
In this paper, we propose to discuss the communication problem in MANET as well as present some characteristic techniques for the design, the analysis and the performance evaluation of distributed communication protocols for mobile ad hoc networks. More specifically, we propose to review two different design techniques. While the first type of protocols tries to create and maintain routing paths among the hosts, the second set of protocols uses a randomly moving subset of the hosts that acts as an intermediate pool for receiving and delivering messages. We discuss the main design choices for each approach, along with performance analysis of selected protocols.

1 Introduction

With the recent dramatic progress in computer platforms and wireless communications, mobile ad hoc networks (MANET) are about to enter the mainstream without however having created yet a solid algorithmic basis for coping with a new set of crucial issues and inherent trade-offs. Critical features, including mobility, severe limitations in power and communication capabilities and limited reliability, have to be carefully taken into account towards designing efficient and robust algorithms for such environments. The

* Azzedine Boukerche's work was partially supported by the Canada Research Chair (CRC) Program, the Canada Foundation for Innovation Grant and Ottawa Distinguished Researcher Award #2021722, and Sotiris Nikoletseas's work was partially supported by the IST/FET Programme of the European Union under contract numbers IST-1999-14186 (ALCOM-FT) and IST-2001-33116 (FLAGS).

communication among mobile hosts is one among the many important issues that need to be resolved efficiently before MANET becomes a commodity.

The problem of communication among remote mobile hosts is one of the most fundamental problems in mobile ad-hoc networks and is at the core of many distributed algorithms, such as counting the number of hosts, leader election, data processing just to mention a few. In mobile ad-hoc networks, a node MH_s that wants to communicate with MH_r might not be within communication range, however it could communicate if other hosts lying "in between" them are willing to forward the packets on their behalves.

The simplest way to establish communication in such networks is to perform the basic flooding paradigm. By flooding, we mean a distributed process of broadcasting a packet to all nodes within the transmission radius of the initiating sender and so on, until all nodes in the sender's "connected component" receive the packet. It is evident that such an approach will waste a large portion of the available bandwidth and therefore it is inefficient to use such schemes.

In order to avoid the flooding problem, several protocols have been proposed in the related literature. A possible categorization of these protocols is based on the approach taken to establish a reliable communication. One approach tries to construct new paths and update these paths dynamically between the mobile hosts. The second approach, instead avoids path creation and maintenance by taking advantage of the hosts movement and accidental meetings in the network area.

1.1 The Path Construction and Maintenance Approach

In an attempt to overcome the inefficiency of the flooding protocols, some algorithms have been proposed that try to construct a a dynamic-graph, e.g., connectivity related dynamic data structure that represents the topology of the network by a series of message passing. By taking advantage of such a data structure, messages can be transmitted over a limited number of intermediate hosts that interconnect MH_s with MH_r, also known as *routing paths* or *routes*. Indeed, this approach is common in ad-hoc mobile networks that either cover a relatively small space (i.e., the temporary network has a small diameter in terms of the transmission range), or it is dense (i.e., comprised of thousands of wireless hosts). In such a case, since almost all "intermediate" locations are occupied by some hosts, communication is possible and broadcasting can be efficiently accomplished as outlined in [1].

In [7, 33], the authors propose the Dynamic Source Routing (DSR) protocol, which uses *on-demand route discovery*. There exists several DSR protocol variations. Ecah of the variations tries to optimize the route discovery overhead. Global positioning system information has been used to reduce the message overhead mainly due to the flooding paradigm. In [**?**], the authors propose two modifications to the DSR ad-hoc routing protocol: (1) Use geographic location information for propagating route requests, and (2) Randomly choosing only a subset of neighbors to which to forward a route request message. The results obtained in [**?**] are quite encouraging.

In [34], the authors present the Ad Hoc On Demand Distance Vector routing (or AODV) protocol that also uses a demand-driven route establishment procedure. More recently two mechanisms have been proposed to deal with the congestion control and communication problems within the AODV routing protocol. First, the authors in [3] have investigated a randomized approach of the AODV protocol. Then they have presented a preemptive ad hoc on-demand distance vector routing protocol for mobile ad hoc networks [?, ?]. They have reported a set of simulation experiments to evaluate their performance when compared to the original AODV on-demand routing protocol. The results they have obtained were quite encouraging.

In [28, 29], the authors have presented a Temporally-Ordered Routing Algorithm (TORA), designed to minimize reaction to topological changes by localizing routing-related messages to a small set of nodes near the change. [30, 19] attempt to combine proactive and reactive approaches in the Zone Routing Protocol (ZRP), by initiating route discovery phase on-demand, but limit the scope of the proactive procedure only to the initiator's local neighborhood. Geographic location information using global positining system (GPS) has also been investigated for the ZRP protocol in [?] using a similar approach as in [?]. In [23], the authors have proposed the Location-Aided Routing (LAR) protocol that uses a global positioning system to provide location information to improve the performance of the protocol by limiting the search for a new route in a smaller "request zone".

1.2 The "Support" Approach

In contrast to all such methods, in [13, 9, 12] a new framework is introduced for protocols designed for highly dynamic movement of the mobile users. The idea of the approach is to take advantage of the mobile hosts natural movement by exchanging information whenever mobile hosts meet incidentally. Specifically, the protocol framework proposes the idea of forcing a small subset of the deployed hosts to move as per the needs of the protocol. This set of hosts is called the "support" of the network (denoted by Σ). Assuming the availability of such hosts, the designer can use them suitably *by specifying their motion* in certain times that the algorithm dictates.

This approach may be particularly useful in cases of rapid deployment of a number of mobile hosts, where 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. This small team of fast moving vehicles can also be a collection of independently controlled mobile modules, i.e. robots. This specific approach is similar to [36] that studies the problem of motion co-ordination in distributed systems consisting of such robots, which can connect, disconnect and move around. Having the support moving in a coordinated way, *i.e. as a chain of nodes*, has some similarities to [36].

Definition 1. The subset of the mobile hosts of an ad-hoc mobile network whose motion is determined by a network protocol P is called the *support* Σ of $\mathcal{P}$. The part of P which indicates the way that members of Σ *move and communicate* is called the *support management subprotocol* M_Σ of $\mathcal{P}$.

The scheme follows the general design principle of mobile networks (with a fixed subnetwork however) called the "two tier principle" [22], 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. The 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.

In a recent paper ([24]), Q.Li and D.Rus present a model which has some similarities to the support approach. 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. It considers the propagation of only one message (end to end) each time, in order to be correct. In contrast, the support scheme allows for simultaneous processing of many communication pairs. In their setting, [24] shows optimality of message transmission times.

2 The AODV Protocol

The Ad-Hoc On-Demand Distance Vector (AODV) routing protocol described in [32] builds on the DSDV algorithm ([31, 33]). AODV is an improvement on DSDV because it typically minimizes the number of required broadcasts by creating routes on a demand basis, as opposed to maintaining a complete list of routes as in the DSDV algorithm. AODV is a pure on-demand route acquisition system, since nodes that are not on a selected path do not maintain routing information or participate in routing table exchanges ([32]).

When a source node desires to send a message to some destination node and does not already have a valid route to that destination, it initiates a path discovery process to locate the other node. It broadcasts a route request (`RREQ`) packet to its neighbors, which then forward the request to their neighbors, and so on, until either the destination or an intermediate node with a "fresh enough" route to the destination is located. AODV utilizes destination sequence numbers to ensure that all routes are loop-free and contain the most recent route information. Each node maintains its own sequence number, as well as a broadcast `ID`. The broadcast `ID` is incremented for every `RREQ` the node initiates, and together with the node's `IP` address, uniquely identifies a `RREQ`. Along with its own sequence number and the broadcast `ID`, the source node includes in the `RREQ` the most recent sequence number it has for the destination. Intermediate nodes can reply to the `RREQ` only if they have a route to the destination whose corresponding destination sequence number is greater than or equal to that contained in the `RREQ`.

During the process of forwarding the `RREQ`, intermediate nodes record in their route tables the address of the neighbor from which the first copy of the broadcast packet is received, thereby establishing a reverse path. If additional copies of the same `RREQ` are later received, these packets are discarded. Once the `RREQ` reaches the destination or an intermediate node with a fresh enough route, the destination/intermediate node responds by unicasting a route reply (`RREP`) packet back to the neighbor from which it first received the `RREQ`. As the `RREP` is routed back along the reverse path, nodes along this path set up forward route entries in their route tables which point to the node from which the `RREP` came. These forward route entries indicate the active forward route. Associated with each route entry is a route timer which will cause the deletion of the entry if it is not used within the specified lifetime. Because the `RREP` is forwarded along the path established by the `RREQ`, AODV only supports the use of symmetric links.

Routes are maintained as follows. If a source node moves, it is able to reinitiate the route discovery protocol to find a new route to the destination. If a node along the route moves, its upstream neighbor notices the move and propagates a link failure notification message (an `RREP` with infinite metric) to each of its active upstream neighbors to inform them of the erasure of that part of the route [33]. These nodes in turn propagate the link failure notification to their upstream neighbors, and so on until the source node is reached. The source node may then choose to reinitiate route discovery for that destination if a route is still desired.

3 The Randomized AODV Protocol

The AODV protocol is extended with a drop factor that induces a randomness feature to result in Randomized Ad-Hoc On-Demand Routing (*R-AODV*) protocol [3]. During the route discovery process, every intermediary or router node between the source and the destination nodes makes a decision to either broadcast/forward the RREQ packet further towards the destination or drop it. Before forwarding a RREQ packet, every node computes the drop factor which is a function of the inverse of the number of hop counts traversed by the RREQ packet. This drop factor lies in the range of 0 to 1. Also, the node generates a random number from 0 to 1. If this random number is higher than the drop factor, the node forwards the RREQ packet. Otherwise, the RREQ packet is dropped. Dropping of RREQ packets does not necessarily result in a new route discovery process by the source node. This is due to the fact that the original broadcast by the source node results in multiple RREQ packets via the neighbors and this diffusing wave results quickly in a large number of RREQ packets traversing the network in search of the destination. A major proportion of these packets are redundant due to the fact that in the ideal case, a single RREQ packet can find the best route. Also, a number of these packets diffusing in directions away from the destination shall eventually timeout. Hence, in *R-AODV*, the aim is to minimize on these redundant RREQ packets, or alternatively, drop as much as possible of these redundant RREQ packets. The drop policy is conservative and its value becomes lesser with higher number of hops. As RREQ packets get near the destination node, the chances of survival of RREQ packets is higher. Hence, the first phase of the route discovering process, that is, finding the destination node, is completed as soon as possible and a RREP packet can be transmitted from the destination node back to the source node.

In *R-AODV* [3], the dropping of redundant RREQ packets reduces a proportion of RREQ packets that shall never reach the destination node, resulting in a decrease of network congestion. Hence, the ratio of the number of packets received by the nodes to the number of packets sent by the nodes, namely, the throughput, should be higher in *R-AODV* compared to AODV.

The following algorithm has been used in the decision making process of whether to drop the RREQ packets by the intermediary or routing nodes (see [3] for more information).

```
Step 1: Calculate drop_factor
        drop_factor = (1/(Hop_count_of_RREQ_packet + 1))
Step 2: Calculate a random value in the range of 0 to 1.
Step 3: If (random_value > drop_factor)
        then broadcast/forward RREQ_packet
        else drop RREQ_packet
```

4 Adaptive Preemptive AODV Protocol

In this section, we review the preemptive protocol first, as described in [?, ?], then we discuss how this preemptive protocol is added into the original AODV.

The preemptive protocol initiates a route rediscovery before the existing path breaks. It overlaps the route discovery routine and the use of the current active path, thereby reducing the average delay per packet. During the development of P_rAODV, we have investigated several preemptive mechanisms. In [?, ?], the following two approaches have been investigated:

(i) Schedule a rediscovery in advance: In this approach, when a reply packet returns to the source through an intermediate node, it collects the information of the links. Therefore, when the packet arrives at the source, the information about the condition of all links will be known, including the minimum value of the lifetime of the links. Hence, we can schedule a rediscovery $T_{rediscovery}$ time before the path breaks.

(ii) Warn the source before the path breaks: Some mechanisms are needed to take care of finding which path is likely to break. We can monitor the signal power of the arrived packets as follows: when the signal power is below a threshold value, we begin the ping-pong process between this node and its immediate neighbor nodes. This node sends to its neighbors in the upstream a hello packet called *ping*, and the neighboring nodes will respond with a hello packet called *pong*. Such *ping-pong* messages should be monitored carefully. In our approach, when bad packets were received (or we timeout on ping packets) a warning message should be sent back to the source during the monitoring period. Upon receiving a warning message, a path rediscovery routine is invoked.

Warning the source: Recall that a preemptive protocol should monitor the *signal power* of the receiving packet. In our approach, we monitor the transfer time for the hello packets. According to [?], $P_n = \frac{P_0}{r^n}$, where P_0 is a constant for each link and n is also a constant, we can see that the distance, which connects to the transfer time of the packet via each link, has a one-to-one relationship to the signal power. Thus, by monitoring the transfer time of packets via each link, we can identify when the link is going to break.

In our protocol, when a node (destination) receives a real packet other than AODV packets, it sends out a pong packet to its neighbor (source), i.e., where the packet comes from. Note that there is no need to send this pong packet if we can include a field in the real packet to store the current time the packet is sent. However, in the original packet, it doesn't have such a field, hence, we need an additional *pong* packet to get such information. In both ping and pong packets, we include a field containing the current time that the packet is sent. When the source monitoring the transfer time is greater than some specific value c_1, it begins the *ping-pong* process. If more than two pong packets arrived with the transfer time greater than a constant c_2, the node will warn the source of the real packet.

Upon receiving the warning message, the source checks if the path is still active. This can easily be done, since each route entry for a destination associates a value, rt_last_send, which indicates if the path is in use or not. If the path is still in use, then a rediscovery routine is invoked.

Theoretically, these two cases can take care of finding new paths before current paths break. The first case is quite helpful if the data transmission rate is slow. If only one data packet is sent out per time-expired period, then even the current route is still needed. Note that this entry may not have packets to go through it during the next time period. This can happen if the interval between two continuous packets is greater that the timeout period of the current route. The second case can be used to find a new path whenever a warning message is generated. However, those two cases are also based on some assumptions, such as the span of a link which can be estimated by the time out of ping packets and the values of those constants are accurate. Once we break those assumptions, the test results will be unexpected.

5 The SUPPORT Protocols

5.1 The Motion Graph Model

In [13] the authors propose an *explicit* model of motions in the light of the fact that the motions of the hosts are the cause of the fragility of the virtual links. Thus they 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 perform inside this space.

They model the space of motions only combinatorially, i.e. as a graph. Future research is expected to complement this effort by introducing geometry details into the model.

In particular, they abstract the environment where the stations move (in three-dimensional space with possible obstacles) by a *motion-graph* (i.e. neglecting the detailed geometric characteristics of the motion. They 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. They approximate this sphere by a cube tc with volume $\mathcal{V}(tc)$, where $\mathcal{V}(tc) \simeq \mathcal{V}(tr)$. 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 2. 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 $\mathcal{V}(tr)$. Given the transmission range tr, n depends linearly on the volume of space $\mathcal{S}$ regardless of the choice of tc, and $n = O\left(\frac{V(\mathcal{S})}{V(tr)}\right)$.

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 the sequel, tc is taken to be a cube, thus 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.

5.2 The Basic Idea of the Protocols

In simple terms, the support protocols work 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 [37].

5.3 The SNAKE Protocol

The main idea of the protocol proposed in [13, 9, 12] 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. [20]). 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 pairwise 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.

[13] assumes 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 is further justified in the correctness and the efficiency analyses of the protocol in [13], where the basic communication protocol should meet some performance requirements *regardless of the motion of the hosts not in Σ*.

A modification of M^S_Σ proposed in [13] 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^S_3 presented in [13] is to use a forward mechanism that transmits incoming messages (in-transit) to neighboring members of the support (due to P^S_1 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.

5.4 The RUNNERS Protocol

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^R_3. 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 presented in [13], the runners protocol does not use the idea of a (moving) backbone subnetwork as no motion subprotocol P^R_1 is used. However, all communication is still routed through the support Σ and it is expected 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). This superiority of the RUNNERS protocol over the SNAKE protocol has been indeed verified (using large scale simulations) in [10]. Also, the recent work of Dimitriou, Nikoletseas and Spirakis [17] on concurrent random walks of interacting particles on graphs provides an analytical tool for the estimation of communication times in the RUNNERS protocol.

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^R_3 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^R_3 is partially based on the *two-phase commit* algorithm as presented in [25] and works as follows.

Let the members of Σ residing on the same site (i.e., vertex) u of G be $MS^u_1, \ldots, MS^u_j$. 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. It is now possible to remove from S_2 all message receipts with smaller ID than this ID.

Recent research towards an analysis of the RUNNERS protocol. In [17], Dimitriou, Nikoletseas and Spirakis study the problem of information propagation between concurrent random walks. The obtained results can be used in the analysis of the expected communication time of the "RUNNERS" protocol.

In particular, they study the following problem: Consider k particles, 1 red and $k-1$ white, chasing each other on the nodes of a graph G. If the red one catches one of the white, it "infects" it with its color. The newly red particles are now available to infect more white ones. When is it the case that all white will become red?

The authors model this problem by k concurrent random walks, one corresponding to the red particle and $k-1$ to the white ones. The *infection time* T_k of infecting *all* the white particles with red color is then a random variable that depends on k, the initial position of the particles, the number of nodes and edges of the graph, as well as on the structure of the graph.

They develop a set of probabilistic tools that are used to obtain upper bounds on the expected value of T_k for general graphs and important special cases (including cliques and expander graphs). They also evaluate and validate their analytical results by large scale experiments.

A recent AODV - RUNNERS comparison. In [8], the authors focus on the effect of mobility and users' density on the performance of routing protocols. In particular, they choose to investigate the impact on AODV (following the path maintenance approach) and RUNNERS (following the support approach). They have implemented the two protocols and performed a large scale and detailed simulation study of their performance. The main findings are: the AODV protocol behaves well in networks of high user density and low mobility rate, while its performance drops for sparse networks of highly mobile users. On the other hand, the RUNNERS protocol seems to tolerate well (and in fact benefit from) high mobility rates and low densities.

5.5 Special Variations of the SUPPORT Protocol

In [9, 14, 15] the authors introduce two practical models of ad-hoc mobile networks and accordingly variate the "RUNNERS" protocol: a) In *hierarchical* ad-hoc networks, comprised of dense subnetworks of mobile users interconnected by a very fast yet limited backbone infrastructure, a "runners" support is used in each lower level of the hierarchy (i.e. in each dense subnetwork). This hierarchical implementation of the support approach is shown to significantly outperform a simple implementation of it in hierarchical ad-hoc networks. b) In *highly changing* ad-hoc networks, where the deployment area changes in a highly dynamic way and is unknown to the protocol, an adaptive runners protocol is introduced. This protocol instead of using a fixed sized support for the whole duration of the protocol, employs a support of some initial (small) size which adapts to the actual levels of traffic and the (unknown and possibly rapidly changing) network area, by changing its size in order to converge to an optimal size, thus satisfying certain Quality of Service criteria.

6 Analysis of the SNAKE Protocol

Time-efficiency of semi-compulsory protocols (i.e. protocols affecting the motion of *some* of the hosts) 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 [11, 20], in [13] the authors 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. This kind of analysis is proposed 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 [21], [18].

In the light of the above, [13] assumes that any host not belonging in the support, conducts a random walk on the motion graph, independently of the other hosts.

The head of the snake does a *continuous time random walk* on the motion graph $G(V, E)$. This is without loss of generality (since, if it is a discrete time random walk, all results transfer easily, see [2]).

Let 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.

The authors in [13] assume that all random walks are *concurrent* and that there is a global time t, not necessarily known to the hosts.

Note 3. Since the motion graph G is finite and connected, the continuous Markov chain abstracting the random walk on it is automatically time-reversible.

Definition 4. 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 5. For the random walk of any particular host, let $\pi()$ be the stationary distribution of its position after a sufficiently long time.

Let E_π [ ] denote the expectation for the chain started at the stationary distribution π.

The communication time of M^S_Σ 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), the authors in [13] 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) Replace the meeting time of u and Σ by a hitting time, using the following thought experiment:
(b1) Fix the support Σ in an "average" place inside G.
(b2) 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) 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. Let the expected value of this hitting time be $E_\pi T^H_\sigma$.

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

Note that 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 [2], they get the following lemma.

Lemma 6. *[13] For any node σ of any graph H in a continuous-time random walk*

$$E_\pi T^H_\sigma \leq \frac{\tau_2\big(1-\pi(\sigma)\big)}{\pi(\sigma)}$$

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 that 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 Markov Chain corresponding to 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. [27]) 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* [16], ch. 1, p.13), they get the following:

Theorem 7. *[13]*

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

Theorem 8. *[13] The expected communication time of the* SNAKE *protocol is bounded above by the formula*

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

Note that the above upper bound is minimized when k $= \sqrt{\frac{2n}{\lambda_2(G)}}$, a fact also verified by simulations (see [11] and also [13]).

The authors in [13] remark that this upper bound is tight in the sense that by [2], Chapter 3,

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

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 [13] show sharp concentration around the mean degree of σ.

6.1 Robustness

Now, we examine the robustness of the support management subprotocol M_Σ^S and M_Σ^R under single stop-faults. We present two Theorems proved in [13]

Theorem 9. *[13] 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 finite expected time 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. □

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 10. *[13] The support management subprotocol M_Σ^R is resilient to t faults, for any $0 \leq t < k$.*

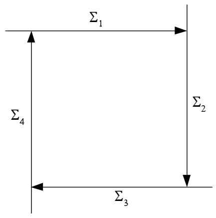

Fig. 1. Deadlock situation when four "snakes" are about to be merged.

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. □

7 Conclusions

In this Chapter we focus on the fundamental problem of communication in mobile ad hoc networks. Towards demonstrating alternative choices in the design of communication algorithms, we reviewed two characteristic approaches in the relevant literature: one that tries to create and maintain routes between mobile hosts, and another that, instead, uses a randomly moving subset of the hosts as "helpers" for receiving and delivering messages. We presented several protocols for each approach, along with a discussion of performance aspects of these protocols.

References

1. M. Adler and C. Scheideler, "Efficient communication strategies for ad-hoc wireless networks," *10th Annual Symposium on Parallel Algorithms and Architectures, ACM*, 1998, pp. 259–268.
2. D. Aldous and J. Fill, "Reversible markov chains and random walks on graphs," http://stat-www.berkeley.edu/users/aldous/book.html, 1999.
3. A. Boukerche, "Congestion control mechanisms for on-demand vector ad hoc routing protocols," *Europar'2003 Conference.*
4. A. Boukerche and L. Zhang "Preemptive on-demand routing protocol for wireless and mobile ad Hoc networks," *IEEE/ACM/SCS 36th Annual Simulation Symposium*, pp. 73-80, 2003.
5. A. Boukerche and L. Zhang, "A performance evaluation of an efficient preemptive on-demand routing protocol for mobile ad hoc networks," *Journal of Wireless and Mobile Communication* (To appear).

6. A. Boukerche, J. Linus, Agrawal Saurabah, "A performance study of dynamic source routing protocols for mobile and wireless ad hoc networks", *EuroPar'2002 Conference*, pp. 957-964, LNCS 2400, 2002.
7. J. Broch, D.B. Johnson, and D.A. Maltz, "The dynamic source routing protocol for mobile ad-hoc networks," *Tech. report, IETF*, Internet Draft, December 1998, draft-ietf-manet-dsr-01.txt.
8. I. Chatzigiannakis, E. Kaltsa, and S. Nikoletseas, "On the effect of user mobility and density on the performance of routing protocols for ad-hoc mobile networks," *Tech. report, CTI*, 2003.
9. I. Chatzigiannakis and S. Nikoletseas, "Design and analysis of an efficient communication strategy for hierarchical and highly changing ad-hoc mobile networks," *ACM/Baltzer Journal of Mobile Networks and Applications (MONET)*, (To appear).
10. I. Chatzigiannakis, S. Nikoletseas, N. Paspalis, P. Spirakis, and C. Zaroliagis, "An experimental study of basic communication protocols in ad-hoc mobile networks," *5th International Workshop on Algorithmic Engineering (WAE 2001)*, Springer-Verlag, 2001, Lecture Notes in Computer Science, LNCS 2141, pp. 159–171.
11. I. Chatzigiannakis, S. Nikoletseas, and P. Spirakis, "Analysis and experimental evaluation of an innovative and efficient routing approach for ad-hoc mobile networks," *4th International Workshop on Algorithmic Engineering (WAE 2000)*, Springer-Verlag, 2000, Lecture Notes in Computer Science, LNCS 1982, pp. 99–110.
12. I. Chatzigiannakis, S. Nikoletseas, and P. Spirakis, "An efficient communication strategy for ad-hoc mobile networks", *15th International Workshop on Distributed Algorithms (DISC 2001)*, Springer-Verlag, 2001, Lecture Notes in Computer Science, LNCS 2180, pp. 285–299.
13. I. Chatzigiannakis, S. Nikoletseas, and P. Spirakis, "Distributed communication algorithms for ad hoc mobile networks," *Journal of Parallel and Distributed Computing (JPDC)* **63** (2003), no. 1, 58–74, Special issue on Wireless and Mobile Ad-hoc Networking and Computing, edited by A. Boukerche.
14. 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.
15. 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.
16. F.R.K. Chung, "Spectral graph theory," *Regional conference series in mathematics* **92** (1994), *CBMS Conference on Recent Advances in Spectral Graph Theory.*
17. T. Dimitriou, S. Nikoletseas, and P. Spirakis, "The infection time of graphs," *Tech. report, CTI*, 2003.
18. Z.J. Haas and M.R. Pearlman, "The performance of a new routing protocol for the reconfigurable wireless networks," *International Conference on Communications, IEEE*, 1998.
19. Z.J. Haas and M.R. Pearlman, "The zone routing protocol (ZRP) for ad-hoc networks," *Tech. report, IETF*, Internet Draft, June 1999, draft-zone-routing-protocol-02.txt.

20. K.P. Hatzis, G.P. Pentaris, P.G. Spirakis, V.T. Tampakas, and R.B. Tan, "Fundamental control algorithms in mobile networks," *11th Annual Symposium on Parallel Algorithms and Architectures, ACM*, 1999, pp. 251–260.
21. G. Holland and N. Vaidya, "Analysis of TCP performance over mobile ad hoc networks," *5th Annual ACM/IEEE International Conference on Mobile Computing (MOBICOM 1999), IEEE/ACM*, 1999, pp. 219–230.
22. T. Imielinski and H.F. Korth, "Mobile computing," Kluwer Academic Publishers, 1996.
23. Y. Ko and N.H. Vaidya, "Location-aided routing (LAR) in mobile ad-hoc networks," *4th Annual ACM/IEEE International Conference on Mobile Computing (MOBICOM 1998), IEEE/ACM*, 1998, pp. 66–75.
24. Q. Li and D. Rus, "Sending messages to mobile users in disconnected ad-hoc wireless networks," *6th Annual ACM/IEEE International Conference on Mobile Computing (MOBICOM 2000), IEEE/ACM*, 2000, pp. 44–55.
25. N.A. Lynch, "Distributed algorithms," Morgan Kaufmann Publishers Inc., 1996.
26. K. Mehlhorn and S. Näher, "LEDA: A platform for combinatorial and geometric computing," Cambridge University Press, 1999.
27. R. Motwani and P. Raghavan, "Randomized algorithms," Cambridge University Press, 1995.
28. V. Park and M. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," *16th Annual IEEE International Conference on Computer Communications and Networking (INFOCOM 1997), IEEE*, 1997, pp. 1405–1413.
29. V.D. Park and M.S. Corson, "Temporally-ordered routing algorithms (TORA) version 1 functional specification," *Tech. report, IETF*, Internet Draft, October 1999, draft-ietf-manet-tora-spec-02.txt.
30. M. Pearlman and Z. Haas, "Determining the optimal configuration for the zone routing protocol," *Journal on Selected Areas in Communications* **17** (2003), no. 8, 1395–1414.
31. C. Perkins, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers," *18th Conference on Communications Architectures, Protocols and Applications (SIGCOMM 1994), ACM*, 1994, pp. 234–244.
32. C. Perkins and E.M. Royer, "Ad-hoc on demand distance vector (AODV) routing," *2nd Annual Workshop on Mobile Computing Systems and Applications, IEEE*, 1999, pp. 90–100.
33. C.E. Perkins, "Ad hoc networking," Addison-Wesley, Boston, 2001.
34. C.E. Perkins and E.M. Royer, "Ad-hoc on demand distance vector (AODV) routing," *Tech. report, IETF*, Internet Draft, September 1999, draft-ietf-manet-aodv-04.txt.
35. C.E. Perkins, E.M. Royer, and S.R. Das, "Ad-hoc on demand distance vector (AODV) routing," *Tech. report, IETF*, Internet Draft, June 2002, draft-ietf-manet-aodv-11.txt.
36. J.E. Walter, J.L. Welch, and N.M. Amato, "Distributed reconfiguration of metamorphic robot chains," *19th ACM Annual Symposium on Principles of Distributed Computing (PODC 2000), ACM*, 2000, pp. 171–180.
37. Y. Zhang and W. Lee, "Intrusion detection in wireless ad-hoc networks," *6th Annual ACM/IEEE International Conference on Mobile Computing (MOBICOM 2000), IEEE/ACM*, 2000, pp. 275–283.

This article was processed using the LaTeX macro package with LLNCS style