An Experimental Study of Basic Communication Protocols in Ad-hoc Mobile Networks *

Ioannis Chatzigiannakis[1,2] Sotiris Nikoletseas[1,2] Nearchos Paspallis[2]
Paul Spirakis[1,2] Christos Zaroliagis[1,2]

[1] Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece
{ichatz,nikole,spirakis}@cti.gr
[2] Department of Computer Engineering and Informatics,
University of Patras, 26500 Patras, Greece
{paspalis,zaro}@ceid.upatras.gr

Abstract. We investigate basic communication protocols in ad-hoc mobile networks. We follow the semi-compulsory approach according to which a small part of the mobile users, the *support* Σ, that moves in a predetermined way is used as an intermediate pool for receiving and delivering messages. Under this approach, we present a new semi-compulsory protocol called the *runners* in which the members of Σ perform concurrent and continuous random walks and exchange any information given to them by senders when they meet. We also conduct a comparative experimental study of the runners protocol with another existing semi-compulsory protocol, called the *snake*, in which the members of Σ move in a coordinated way and always remain pairwise adjacent. The experimental evaluation has been carried out in a new generic framework that we developed to implement protocols for mobile computing. Our experiments showed that for both protocols only a small support is required for efficient communication, and that the runners protocol outperforms the snake protocol in almost all types of inputs we considered.

1 Introduction

Mobile computing has been introduced in the past few years as a new computing environment. Since mobile computing is constrained by poor resources, highly dynamic variable connectivity, and volatile energy sources, the design of stable and efficient mobile information systems is greatly complicated. Until now, two basic models have been proposed for mobile computing. The earlier (and commonly used) model is the *fixed backbone* model which assumes that an existing infrastructure of support stations with centralized network management is provided in order to ensure efficient communication. A more recent model is the *ad-hoc model* which assumes that mobile hosts can form networks without the participation of any fixed infrastructure. An *ad-hoc mobile network* [1, 8] is a collection of mobile hosts with wireless network interfaces forming a temporary network without the aid of any established infrastructure or centralized administration. In an ad-hoc network two hosts that want to communicate may not be within wireless transmission range of each other, but could communicate if other hosts between them are also participating in the ad-hoc network and are willing to forward packets for them.

A usual scenario that motivates the ad-hoc mobile model is the case of rapid deployment of mobile hosts in an unknown terrain, 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 it has become temporarily unavailable (i.e., destroyed or down).

A *basic communication problem* in such ad-hoc mobile networks, is to send information from some *sender* $\mathcal{S}$, to another designated *receiver* $\mathcal{R}$. Note that ad-hoc mobile networks are dynamic in nature,

* This work was partially supported by the IST Programme of EU under contract no. IST-1999-14186 (ALCOM-FT), by the Human Potential Programme of EU under contracts no. HPRN-CT-1999-00104 (AMORE) and HPRN-CT-1999-00112 (ARACNE), and by the Greek GSRT project ALKAD.

in the sense that local connections are temporary and may change as users move. The movement rate of each user might vary, while certain hosts might even stop in order to execute location-oriented tasks (e.g., take measurements). In such an environment, executing a distributed protocol has certain complications, since communication between two hosts may be a highly non-trivial task.

The most common way to establish communication is to form paths of intermediate nodes (i.e., hosts), where it is assumed that there is a link between two nodes if the corresponding hosts lie within one another's transmission radius and hence can directly communicate with each other [5, 13–15]. Indeed, this approach of exploiting pairwise communications is common in ad-hoc mobile networks that either cover a relatively small space (i.e., the temporary network has a small diameter with respect to the transmission range), or are dense (i.e., thousands of wireless nodes). Since almost all locations are occupied by some hosts, broadcasting can be efficiently accomplished.

In wider area ad-hoc networks however, broadcasting is impractical, as two distant peers will not be reached by any broadcast since users do not occupy all intervening locations, i.e., a sufficiently long communication path is difficult to establish. Even if such a 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 the 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 nodes.

A different approach to solve this basic communication problem is to take advantage of the mobile hosts natural movement by exchanging information whenever mobile hosts meet incidentally. When the users of the network meet often and are spread in a geographical area, flooding the network will suffice. 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 care of such situations.

One way to alleviate these problems is to force mobile users to move according to a specific scheme in order to meet the protocol demands. Our approach is based on the idea of forcing only a very small subset of mobile users, called the *support* Σ of the network, to move as per the needs of the protocol. Such protocols are called *semi-compulsory* protocols.

The first semi-compulsory protocol for the basic communication problem was presented in [2]. It uses a snake-like sequence of support stations that always remain pairwise adjacent and move in a way determined by the snake's head. The head moves by executing a random walk over the area covered by the network. We shall refer to this protocol as the *snake protocol*. The snake protocol is theoretically analyzed in [2] using interesting properties of random walks and their meeting times. A first implementation of the protocol was developed and experimentally evaluated in [2] with the emphasis to confirm the theoretical analysis. Both the experiments and the theoretical analysis indicated that only a small support is needed for efficient communication.

In this paper we firstly present a new semi-compulsory protocol for the basic communication problem studied here. The new protocol is based on the idea that the members of Σ move like "runners", i.e., they move independently of each other sweeping the whole area covered by the network. When two runners meet, they exchange information given to them by senders encountered. We shall refer to this protocol as the *runners protocol*. The new protocol turns out to be more robust than the snake protocol. The latter is resilient only to one fault (one faulty member of Σ), while the former is resilient to t faults for any $0 < t < |\Sigma|$.

The second contribution of this paper is a comparative experimental study of the snake and the runners protocols based on a new generic framework that we developed to implement protocols for mobile computing. Based on the implementation in [2], we have redesigned the fundamental classes and their functionality in order to provide this generic framework that allows implementation of any mobile protocol. Under this framework, we have implemented the runners protocol and re-implemented the snake protocol. All of our implementations were based on the LEDA platform [11]. We also have extended the experimental setup in [2] to include more pragmatic test inputs regarding motion graphs, i.e., graphs which model the topology of the motion of the hosts. Our test inputs included both random as well as more structured graphs. In conducting our experimental study, we were interested in providing measures on communication times (especially average message delay), message delivery rate, and support utilization (total number of messages contained in all members of the support).

Our experiments showed that: (i) for both protocols only a small support is required for efficient communication; (ii) the runners protocol outperforms the snake protocol in almost all types of inputs we considered. More precisely, the runners protocol achieve a better average message delay in all test inputs considered, except for the case of random graphs with a small support size. The new protocol achieves a higher delivery rate of messages right from the beginning, while the snake protocol requires some period of time until its delivery rate stabilizes to a value that is always smaller than that of runners. Finally, the runners protocol has smaller requirements for the size of local memory per member of the support.

Previous Work: A similar approach is presented in [6] where a *compulsory* protocol is introduced in the sense that all users are forced to perform concurrent and independent random walks on the area covered by the network.

A model similar to the above is presented in [10]. The protocol forces all mobile users to slightly deviate (for a short period of time) from their predefined, deterministic routes, in order to propagate the messages. This protocol is also compulsory for any host and it works only for deterministic host routes.

Adler and Scheideler [1] in a previous work, dealt only with *static* transmission graphs, i.e., the situation where the positions of the mobile hosts and the environment do not change. In [1] 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.

2 The Model of the Space of Motions

We use the graph theoretic model introduced in [6] and also used in [2] that maps the movement of the mobile users in the three-dimensional space S to a so-called *motion graph* $G = (V, E)$ (its precise definition will be given shortly). Let $|V| = n$, and $|E| = m$.

The environment where the stations move (in three-dimensional space with possible obstacles) is abstracted by a graph by neglecting the detailed geometric characteristics of the motion (we expect that future research will incorporate geometry constraints into the subject). We first assume that each mobile host has a transmission range represented by a sphere tr having the mobile host as its center. This means that any other host inside tr can receive any message broadcasted by this host. We approximate this sphere by a cube tc with volume $\mathcal{V}(tc)$, where $\mathcal{V}(tc) < \mathcal{V}(tr)$. The size of tc can be chosen in such a way that its volume $\mathcal{V}(tc)$ is the maximum integral value 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 S, S is quantized into cubes of volume $\mathcal{V}(tc)$.

The *motion graph* $G = (V, E)$, which corresponds to the above quantization of S, is constructed in the following way. There is a vertex $u \in V$ representing a cube of volume $\mathcal{V}(tc)$. Two vertices u and v are connected by an edge $(u, v) \in E$ if the corresponding cubes are adjacent.

Let $\mathcal{V}(S)$ be the volume of space S. Clearly, $n = V(S)/V(tc)$ and consequently n is a rough approximation of the ratio $V(S)/V(tr)$. Since the edges of G represent neighbouring polyhedra each node is connected with a constant number of neighbours, which yields $m = \Theta(n)$. In our case where tc is a cube, G has maximum degree 6 and hence $m \leq 6n$. This implies that the motion graph G is a bounded degree graph.

3 Description of the Implemented Protocols

We start with a few definitions that will be used in the description of the protocols. As mentioned before, we are interested in ad-hoc networks that allow a small part of their hosts to serve the network needs. The class of ad-hoc mobile network protocols which enforce a (small) subset of the mobile hosts to move in a certain way is called the class of *semi-compulsory* protocols.

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 P. The part of P which indicates the way in which the members of Σ move and communicate is called the *support management subprotocol* M_Σ of P.

In addition, we may wish that the way hosts in Σ move and communicate can tolerate failures of hosts. In such a case, the protocol is called *robust*. A protocol is called *reliable* if it allows the sender to be notified about delivery of the information to the receiver.

We assume that the motions of the mobile users which are not members of Σ are arbitrary but *independent* of the motion of the support (i.e., we exclude the case where some of the users not in Σ are deliberately trying to avoid Σ). This is a pragmatic assumption usually followed by application protocols.

In the following, we assume that message exchange between nodes within communication distance of each other takes negligible time (i.e., the messages are short packets and the wireless transmission is very fast). We further assume that all users (even those not in the support) perform independent and concurrent random walks. In fact, the assumption that the mobile users move 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 [5, 7].

We are now ready to describe the two protocols, each one having a different support management subprotocol M_{Σ}.

3.1 The Snake Protocol

The main idea of the protocol proposed in [2] is as follows. There is a set-up phase of the ad-hoc network, during which a predefined number, k, of hosts, become the nodes of the support. The members of the support perform a leader election, which 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 list of k nodes), so that they sweep (given some time) the entire motion graph. Their motion is accomplished in a distributed way via a *support motion subprotocol* P_1. Essentially the motion subprotocol P_1 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". When some node of the support is within the 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 in every node of the support using a *synchronization subprotocol* P_3. When a receiver comes within the 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. Duplicate copies of the message are then removed from the other members of the support.

In this protocol, the support Σ plays the role of a (moving) backbone subnetwork (of a "fixed" structure, guaranteed by the motion subprotocol P_1), through which all communication is routed. The theorerical analysis in [2] shows that the average message delay or communication time of the snake protocol is bounded above by the formula $\frac{2}{\lambda_2(G)}\Theta(n/k) + \Theta(k)$ where G is the motion graph, $\lambda_2(G)$ is its second eigenvalue, n is the number of vertices in motion graph G, and $k = |\Sigma|$. More details for the snake protocol can be found in [2]. It can be also proved (see [4]) that the snake protocol is reliable and partially robust (resilient to one fault).

3.2 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 pairwise adjacency between members of Σ, all hosts sweep the area by moving independently from each other. When two runners meet, they exchange any information given to them by senders encountered using a new synchronization subprotocol P_3'. As in the snake case, when some node of the support is within the communication range of a sender, the underlying sensor sub-protocol P_2 notifies the sender that it may send its message(s). When a user comes within the communication range of a node of the support which has a message for the designated receiver $\mathcal{R}$, the waiting messages are forwarded to the receiver.

The runners protocol does not use the idea of a (moving) backbone subnetwork as no motion subprotocol P_1 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 reality, 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 site of the motion graph G, the synchronization subprotocol P_3' is activated. The subprotocol imposes that when runners meet on the same site, 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' is partially based on the *two-phase commit* algorithm as presented in [12] 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' 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 is to construct, for each sender, an ordered list of message IDs contained in S_2. This ordered sequence of IDs will have gaps – some messages will still have to be delivered, and thus not part of S_2. In this list, we can identify the maximum ID before the first gap and remove from S_2 all message receipts with smaller ID. Although this is a rather simple approach, our experiments showed that it effectively reduced the memory usage.

The described support management subprotocol is clearly reliable. It is also robust as it is resilient to t faults, for any $0 \leq t < k$. This can be 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.

4 Implementation and Experimental Results

4.1 Implementation Details

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 [11]. 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 behaviours. 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.

To extend our previous implementation [2], we have redesigned the fundamental classes and their functionality in order to provide a generic framework for the implementation of any mobile protocol. Our implementation of ad-hoc mobile networks is now based on a new base class called `mobile_host`. This class is then extended to implement the class `mh_user` that models a user of the network that acts as sender and/or receiver, and the `mh_snake` and `mh_runner` classes that implement the support management subprotocols respectively.

Also in our previous implementation we had a class called `transmission_medium` to realize the exchange of messages between mobile hosts. In the new implementation, this class has been replaced by a so-called `environment` class which handles not only message exchange, but also the movement of mobile hosts (which can follow various motion patterns).

We have also augmented the implementation of the original algorithm in [2] with additional procedures that speed up execution time. For example, based on the assumption that all mobile hosts use the same communication equipment with fixed transmission range, we are able to implement efficiently the simulated environment by precomputing all possible neighbourhoods of fixed diameter, which allows us to easily abstract the underlying sensor subprotocol P_2.

In order to model the memory capabilities of the members of the support and the users of the network, we used the template `set` class that LEDA provides. This allows us to abstract the storage mechanisms of a host and focus on the other (and more interesting) aspects of the implementation.

It is worth noting that in [2] we did not implement the synchronization subprotocol P_3. This did not affect the behaviour of the protocol, but helped us avoiding to further complicate the implementation. However, in order to provide a fair comparison between the two different support management subprotocols (i.e., the snakes and runners), we have implemented in this work the subprotocol P_3 for the snake protocol, and we also counted the extra delay that was imposed by the synchronization of the mobile support hosts.

4.2 Experimental Setup

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 five kinds of inputs, unstructured (random) and more structured ones. Each kind of input corresponds to a different type of motion graph. We started with the graph families considered in [2], namely random graphs, 2D grid graphs, and bipartite multi-stage graphs. We extended the above experimental setup by considering also 3D grid graphs and two-level motion graphs that are more close to pragmatic situations.

We considered several values for n in the range $[400, 6400]$ and different values for the support size $k = |\Sigma|$, namely $k \in [3, 45]$. 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. Each experiment we carried out, proceeds in lockstep rounds called *simulation rounds* (i.e., we measure simulation time in rounds). Each mobile host h is considered identical in computing and communication capability. During a simulation round, h moves to an adjacent vertex of the motion graph G and performs some computation. If $h \in \Sigma$, then its move and local computation are determined by the snake or the runners protocol. If $h \notin \Sigma$, then its move is random and with probability $p = 0.01$ the host h generates a new message (i.e., a new message is generated roughly every 100 rounds) by picking a random destination host. The selection of this value for p 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, 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 $n \in \{1600, 3200, 6400\}$ 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.

3D Grid Graphs. To evaluate the performance of our protocols over 3D space, we considered 3D grid graphs $(n^{1/3} \times n^{1/3} \times n^{1/3})$ to model the motion of hosts in 3D space. We used three different such graphs with $n \in \{512, 1000, 1280\}$ over different values of $k \in [3, 45]$.

Bipartite multi-stage graphs. A bipartite multi-stage graph is a graph consisting of a number of stages (or levels) ξ. 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 they 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 choose $p = \frac{\xi}{n} \log \frac{n}{\xi}$ 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 or 9 stages, number of vertices $n \in$\{1600, 3200, 6400\}, and different values of $k \in [3, 45]$.

Two-level motion graphs. Motivated by the fact that most mobile users usually travel along favourite routes (e.g., going from home to work and back) that usually comprise a small portion of the whole area covered by the network (e.g., urban highways, ring roads, metro lines etc.), and that in more congested areas there is a high volume of user traffic (e.g., city centers, airport hubs, tourist attractions, etc.), we have considered a two-level motion graph family to reflect this situation (this family has some similarities with the two-level semirandom graphs considered in [9, 16] for completely different problems and settings).

Let $d(u, v)$ denote the distance between vertices u and v in G, i.e., the length of a shortest path joining them. A *two-level graph* consists of subgraphs of the motion graph representing congested areas that are interconnected by a small number of paths representing the most favourite routes among them. A subgraph G_c is defined by randomly selecting a vertex u of G and then declare as vertices of G_c those vertices $v \in G$ with $d(u, v) \leq c$, where c is a constant denoting the diameter of the congested area. Let f be the number of subgraphs defined. The paths representing favourite routes are specified as follows: temporarily replace each G_c by a supervertex, find shortest paths among supervertices, and finally select those edges that belong to shortest paths. If we end up with more than αf paths, where $\alpha > 1$ is a constant, then we either arbitrarily select αf of them, or find a minimum spanning tree that spans the supervertices and use its edges as the selected ones. The majority of hosts is forced to move either within the subgraphs (congested areas) or along the paths connecting subgraphs (favourite routes).

We constructed a number of different data sets by tuning the various parameters, i.e., the diameter c of the congested areas and the number f of selected vertices (number of subgraphs representing congested areas). Note that for each message generated, the destination is chosen at random among all users of the network (i.e., the sender-receiver pairs are not fixed).

4.3 Experimental Results

We measured the total delay of messages exchanged between pairs of sender-receiver users. 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 five different test inputs we considered are illustrated in Figures 1–5. 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 [2]. 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 behaviour of the snake protocol is similar to the one reported in [2], although we count here the extra delay time imposed by the 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 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 1) and bipartite multi-stage graphs (Figure 4), but is substantially better in the more structured cases (grid and two-level graphs; cf. Figures 2, 3, and 5). 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 [2] (see also Section 3.1) which is apparently larger than the corresponding constant of the runners protocol.

An interesting observation concerns the case of 2-level graphs (Figure 5), where the results look similar to those of grid graphs. Interestingly, for this case where the users do not perform random walks (as in the grid case) but their motion is getting more restricted (though still arbitrary), the performance characteristics of both protocols remain unchanged. Hence, we may conclude that for the case of more structured inputs the assumption of having the users performing continuous random walks over the motion graph G does not seem to affect performance. We believe that it is challenging to incorporate other sophisticated mechanisms to the support management subprotocols in order to take advantage of such user behaviour patterns.

In contrast to the experiments carried out in [2], in this paper we also consider the utilization of the support protocol, i.e., the total number of multiple message copies stored in the support structure at any given time, as well as the message delivery rate.

In Figure 6 the total number of message copies generated by each protocol is shown for the first 7000 rounds of simulation on a bipartite multi-stage graph with $n = 6400$ and $k = 15$ (similar results hold for other test inputs and sizes). We observe that the snake protocol generates initially less copies that the runners. As the simulation goes on, the rate of generating redundant messages by the snake protocol increases faster than that of the runners. This can be partially justified by considering the average message delay. In the case of the snake, the overall message delay is worse than that of the runners, which implies that it will take longer to meet a sender. In addition, each message (and its copies) will remain for a longer period within the support structure of the snake until the designated receiver $\mathcal{R}$ is encountered. Therefore, at the beginning the snake protocol generates less copies, as less messages have been received, while as the simulation goes on, the total number of redundant copies increases as more messages are still pending. This observation implies that the runners protocol utilizes more efficiently the available resources as far as memory limitations are concerned.

Another way to evaluate the performance of the two protocols is to consider the message delivery rate. In Figure 7 the overall delivery rate of messages (in percentages) is projected over the simulation time (in rounds) for a 3D grid graph with $n = 6400$ and $k = 15$ (similar results hold for other test inputs and sizes). The snake protocol is slower in delivering messages than the runners protocol for the same reason explained above regarding utilization. As the simulation time increases, the two protocols finally reach high levels of delivery rates. However, it is clear that the runners will reach these high levels much faster compared to the snake. This provides an explanation regarding the total number of message copies stored within the support members of the snake protocol. Recall that the support synchronization subprotocol P_3 will need $O(k)$ time to generate k copies for each message, thus as the delivery rate of the snake is initially low, the total number of message copies will also be low. As the rate increases over time, P_3 will generate more message copies and thus increase the total number of message copies stored within the support structure.

5 Closing Remarks and Future Work

We have presented a new protocol for a basic communication problem in an ad-hoc mobile network and we provided a comparative experimental study of a new protocol (runners) and of a new implementation of an existing one (snake). Our experiments showed that the new protocol outperforms the previous one in almost all test inputs we considered.

There are several directions for future work. An obvious one is to analyze theoretically the runners protocol. Another one is to enforce a specific motion pattern to the support (e.g., moving along a spanning subgraph of the motion graph) instead of doing random walks. The reason that the head of the snake, or the runners perform a random walk is to ensure that (within some finite time) the whole motion graph is covered, without any need for memory (other than local) that could be used to provide topology details.

In the runners setting, an interesting extension would be to introduce the notion of births and deaths on the population Σ. Whenever a runner jumps to a site which is already occupied by another member of Σ, it is removed from the support after transmitting S_1 and S_2. Similarly, when a host meets no member of Σ for some period of time, then with some probability it is announced member of the support. The resulting support management subprotocol is still semi-compulsory, although it is more adaptive to the user needs and movement behaviour.

We have not addressed experimentally the issue of robustness. We note that the runners are resilient to t failures, for any $0 \leq t < k$, however we have neither tried to evaluate the performance in such cases nor provide a proof of correctness. In the case of the snake protocol, resilience to more than one faults is difficult to achieve, since when a fault occurs within the members of the support, the snake (or a part of it if some fault has already occurred) is split into two segments. Providing a mechanism that will merge the possibly several segments when they meet again is not so easy as deadlocks might occur rendering the support useless. Since mobile networks are based on volatile resources, the robustness aspects of the proposed schemes must be further investigated.

Finally, we plan to incorporate in our generic implementation framework hierarchical routing protocols for mobile networks [3] that will use the snake or the runners protocols for basic communication in the lower layers of the hierarchy.

References

1. 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.
2. 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.

3. 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 IPDPS'01, 2001.
4. I. Chatzigiannakis, S. Nikoletseas, and P. Spirakis. Self-Organizing Ad-hoc Mobile Networks: The problem of end-to-end communication. To appear as a short paper In *Proc. 20th Annual Symposium on Principles of Distributed Computing* – PODC'01, 2001.
5. Z. J. Haas and M. R. Pearlman. The performance of a new routing protocol for the reconfigurable wireless networks. In *Proc. ICC'98*, 1998.
6. 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 Symposium on Parallel Algorithms and Architectures* – SPAA'99, 1999.
7. 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.
8. T. Imielinski and H. F. Korth. *Mobile Computing*. Kluwer Academic Publishers, 1996.
9. 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 2000, pp. 59–78.
10. 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.
11. K. Mehlhorn and S. Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999.
12. N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., 1996.
13. 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, Oct. 1999.
14. 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.
15. 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.
16. E.W. Zegura. Modeling Topology of Large Internetworks. http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html

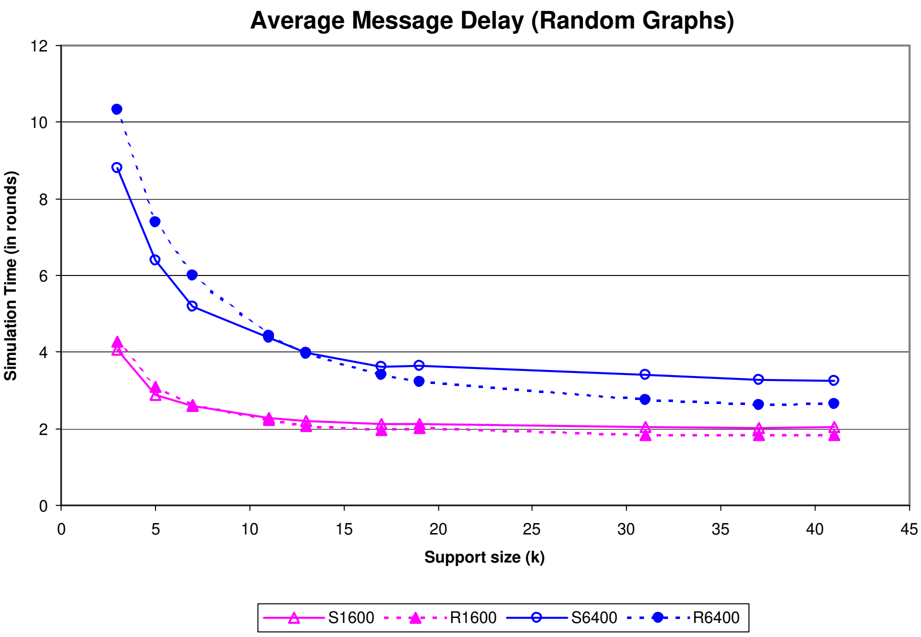

Fig. 1. Average message delay over random graphs with different sizes (n) and varied support size k.

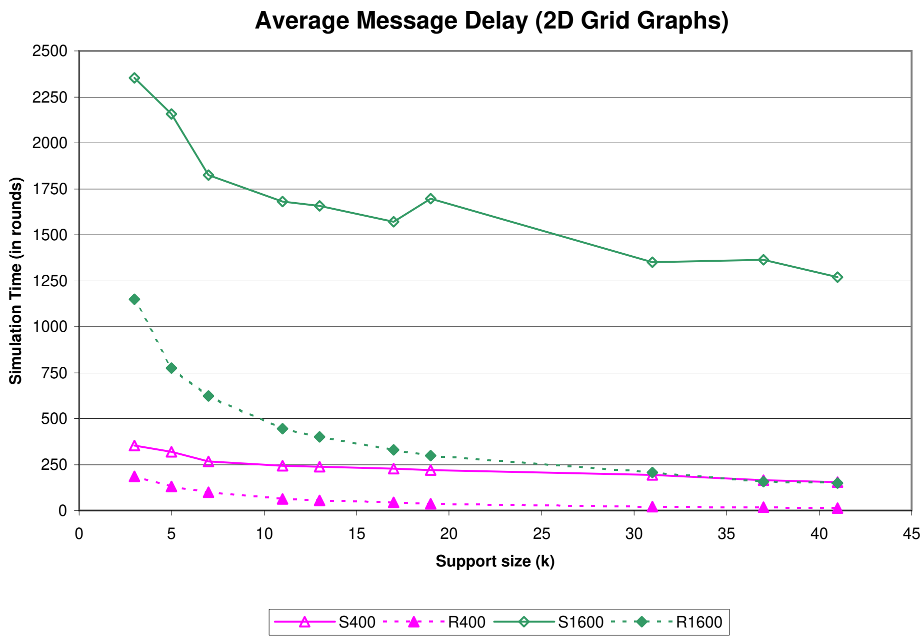

Fig. 2. Average message delay over 2D grid types of motion graphs with different sizes (n) and varied support size k.

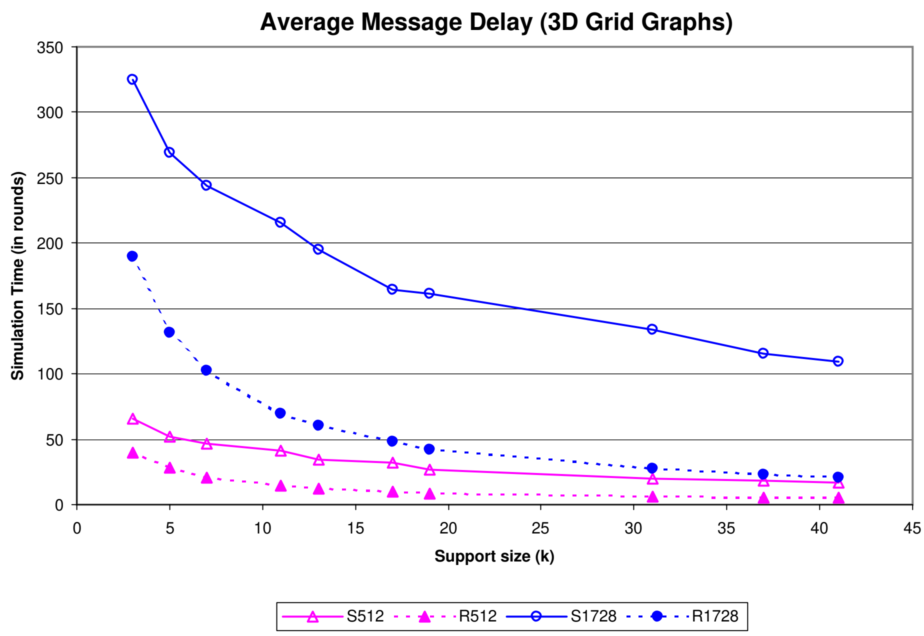

Fig. 3. Average message delay over 3D grid types of motion graphs with different sizes (n) and varied support size k.

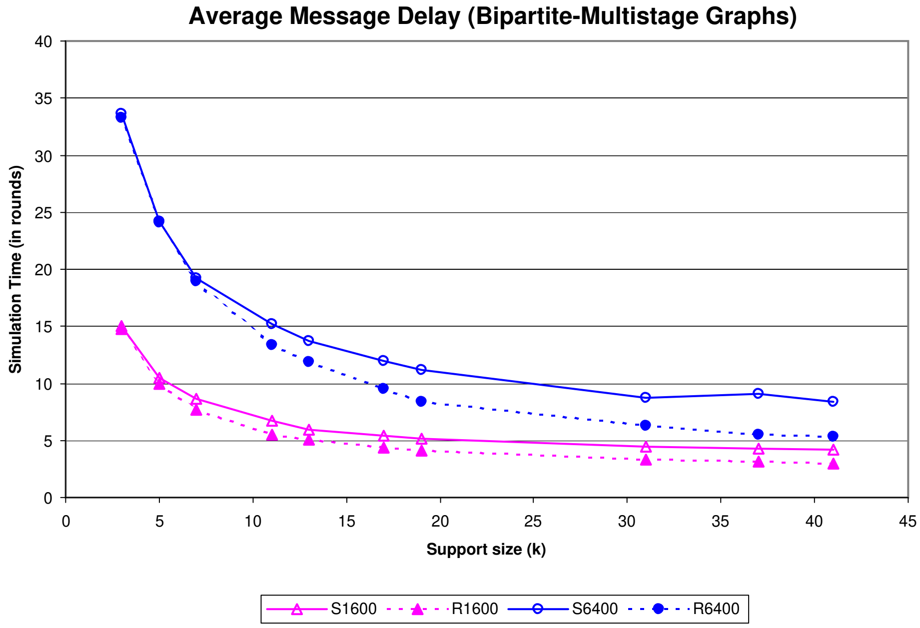

Fig. 4. Average message delay over bipartite multi-stage motion graphs with different sizes (n) and varied support size k.

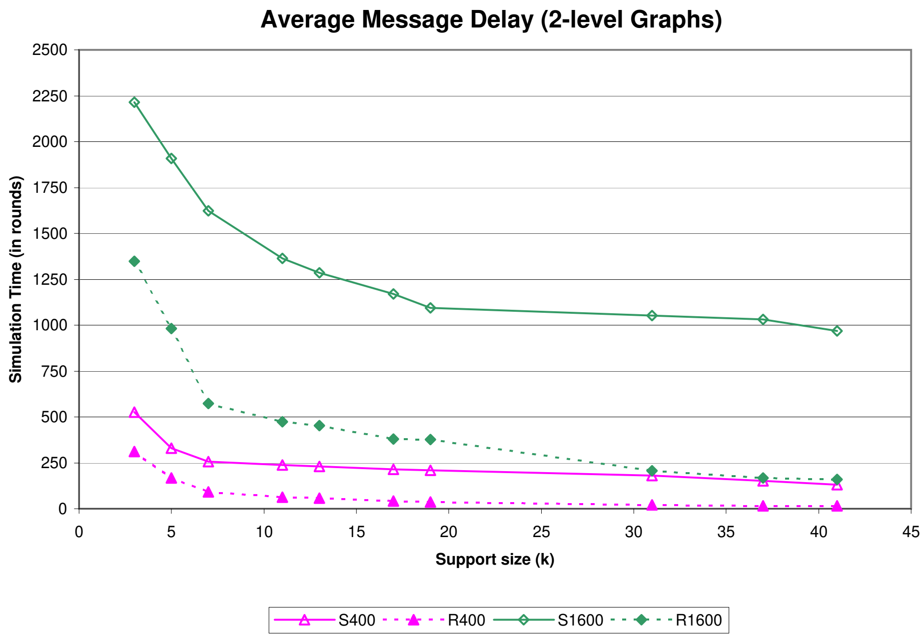

Fig. 5. Average message delay over 2-level semi-random motion graphs with different sizes (n) and varied support size k.

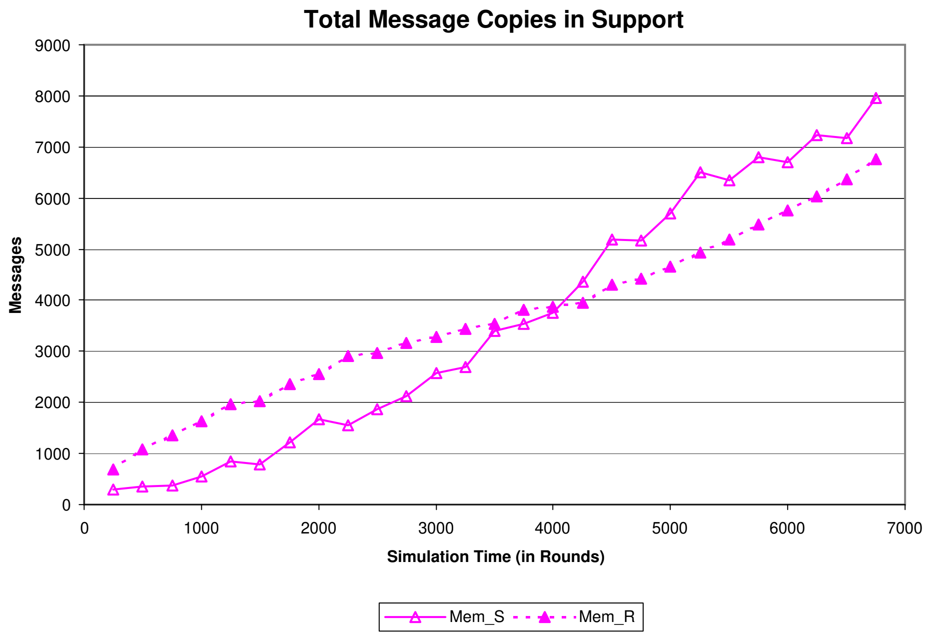

Fig. 6. Total number of message copies within the support structure over simulation time (in rounds).

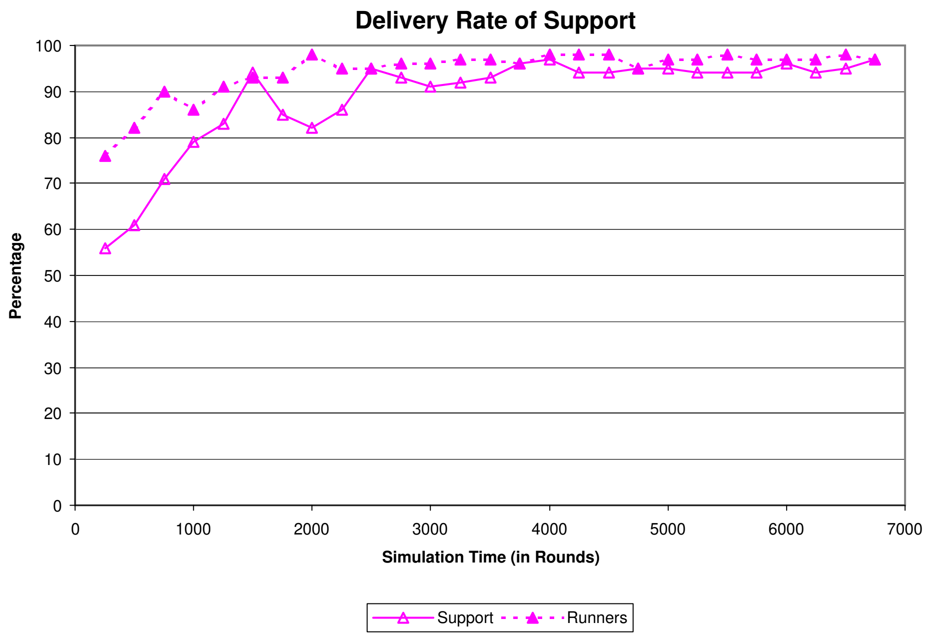

Fig. 7. Message Delivery rate of support over simulation time (in rounds).