Experimenting with Relay Protocols for Communication in Ad-hoc Mobile Networks *

Ioannis Chatzigiannakis† Sotiris Nikoletseas† Paul Spirakis† Christos Zaroliagis†

April 15, 2002

Abstract

We investigate the efficiency of protocols for basic communication in ad-hoc mobile networks. All protocols are classified in the semi-compulsory, relay category according to which communication is achieved through a small team of mobile hosts, called the *support*, which move in a predetermined way and serves as an intermediate pool for receiving and delivering messages. We implement a new semi-compulsory, relay protocol in which the motion of the support is based on a so-called "hunter" strategy developed for a pursuit-evasion game. We conduct an extensive, comparative experimental study of this protocol with other two existing protocols, each one possessing a different motion for its support. We considered several types of inputs, including among others two kinds of motion patterns (random and adversarial) for the mobile hosts not in the support. Our experiments showed that for all protocols the throughput scales almost linearly with the number of mobile hosts in the network, and that a small support size suffices for efficient communication. An interesting outcome is that in most cases the new protocol is inferior to the other two, although it has a global knowledge of the motion space.

1 Introduction

Mobile computing has been recently introduced as a new computing environment [14]. Designing stable and efficient mobile information systems is a highly complex task, since mobile computing is constrained by poor resources, highly dynamic variable connectivity, and volatile energy sources. Mobile computing has been studied under two basic models. The earlier (and commonly used) is the *fixed backbone* model which assumes that an 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, 14, 19] is a collection of mobile hosts (or users) 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 that lie between them 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).

*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).

†Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece, and Dept of Computer Engineering and Informatics, University of Patras, 26500 Patras, Greece. Emails: {ichatz,nikole,spirakis,zaro}@cti.gr

One of the most important problems in an ad-hoc network is to send a piece of information from some mobile host (sender $\mathcal{S}$) to another host (receiver $\mathcal{R}$). We shall refer to this as the *basic communication problem*. Ad-hoc mobile networks are highly dynamic in nature, in the sense that local connections are temporary and may change as hosts move. The movement rate of each host might vary, while certain hosts might even stop in order to execute location-oriented tasks (e.g., take measurements). In such a highly changing environment, executing a distributed protocol has certain extra complications, since communication between hosts may be a highly non-trivial task.

The most common way to establish communication is through the *path construction* approach, i.e., to form paths of intermediate hosts where two hosts are assumed to be connected (by a virtual link) if one host lies within the other's transmission range [19]. 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 with respect to the transmission range), or are dense (i.e., comprised of thousands of wireless hosts). In such cases, since almost all locations are occupied by some hosts, communication is possible and broadcasting can be efficiently accomplished [1].

In wider area ad-hoc networks however, broadcasting may become impractical, as two distant peers may not be reached by any broadcast since hosts may not occupy all intervening locations. Even if a (sufficiently long) communication path is established, single link failures happening when a small number of hosts 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. In addition, even if mobile hosts are almost immobile (so that the established communication paths are not destroyed), it was recently shown in [9, 16] that the *capacity* or *throughput* (average number of messages delivered per unit of time) turns out to be a bottleneck as it is proportional to $O(1/\sqrt{u})$, where u is the total number of mobile hosts. These results suggest that the path construction approach for achieving communication in ad hoc networks is inherently non-scalable.

A different approach to overcome the above performance limitations and solve the basic communication problem is to take advantage of the mobile hosts natural movement and transmit only when the source and destination hosts meet incidentally (come within each other's transmission range). Protocols based on this idea are divided into *non-compulsory* and *compulsory*. A *non-compulsory* protocol is one whose execution does not affect the movement of the mobile hosts. When the hosts of the network meet often and are spread over a small geographical area, flooding the network will suffice. It is evident, however, that if the hosts are spread in remote areas and they do not move beyond these areas, then there is no way for information to reach them, unless the protocol takes care of such situations. One way to address these problems is to force mobile hosts to move according to a specific scheme in order to meet the protocol demands, thus yielding the so-called *compulsory* protocols. Such protocols (see e.g., [11, 17]) request that all mobile hosts perform certain moves in order to guarantee correctness of the protocol.

A compromise between the above two extremes is the *semi-compulsory* protocols, introduced in [3]. The idea is to deploy a small team of mobile hosts, called the *support* Σ of the network, and force this small team to move as per the needs of the protocol (the move of the rest is not affected by the protocol). Each mobile host that wants to send some information to another host sends its message to any member σ of Σ, whenever σ comes within its transmission range. In addition, during this communication, it can receive any messages that have designated it as the receiver. Hence, the support serves as a set of mobile *relay* hosts, that temporarily buffer messages until they get delivered. The expectation is that as this small team of mobile hosts moves randomly around the area of the ad-hoc network, it will cover the entire area in sufficiently short time, thus servicing effectively all messages. Because of the relay of messages to be delivered within Σ, we shall refer to such protocols as *relay protocols*.

The first semi-compulsory, relay protocol for the basic communication problem was presented and implemented in [3]. It is called the *snake protocol*, since it uses a snake-like structure of support hosts that always remain pairwise adjacent and move in a way determined by the snake's head. The head moves by executing a random walk in the area covered by the network. The theoretical analysis and its experimental evaluation in [3] study the average case performance of the protocol and assume that the mobile hosts perform random walks.

A second semi-compulsory, relay protocol for the basic communication problem was presented in [4]. It is called the *runners protocol*, since it 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 any information given to them by any senders encountered. In [4], a comparative experimental study was conducted between the snake and the runners protocols, based on a new implementation environment and on an extensive experimental setup that included both random as well as pragmatic test inputs regarding *motion graphs*, i.e., graphs which model the topology of the motion of the mobile hosts. For the case where the mobile hosts perform concurrent and independent random walks and for almost all inputs considered, the runners protocol outperformed the snake protocol.

A common characteristic of the snake and runners protocols is that they do not require any global knowledge of the motion space. We shall refer to such protocols as *agnostic* protocols. On the other hand, we shall call *gnostic* those protocols whose correct execution requires global knowledge of the motion space. Agnostic protocols try to make decisions based only on local knowledge. Gnostic protocols maintain global knowledge of the motion space and are thus capable of making more accurate decisions based on the way mobile hosts move in the motion space. Thus, gnostic protocols make stronger assumptions that may lead to stronger results, while agnostic ones are potentially less efficient, though more general.

The contribution of this paper is twofold. Firstly, we have implemented a new semi-compulsory, relay protocol which is based on a search strategy for the pursuit-evasion game on graphs given recently in [2]. This protocol is a gnostic one and we shall call it the *hunters protocol*, since it is based on the idea of the "hunter" strategy developed in [2] by which a "hunter" is trying to catch a "rabbit" in the motion space graph. We generalize this idea and deploy various hunters (members of Σ) which follow the "hunter" strategy in order to deliver (and receive) messages to (from) the mobile hosts. Secondly, we performed an extensive, comparative experimental study of all the above protocols using and considerably extending the experimental setup developed in [4]. We were interested in providing measures on communication time (average message delay), memory usage of Σ (both average and maximum per support member), and throughput (the latter two were not considered in [4]). We considered various message generation rates with which the mobile hosts generate new messages (only a single message generation rate was considered in [4]), and an adversarial motion pattern for those hosts not in Σ (only random motion patterns were considered in [4]). The latter is based on an adversarial "rabbit" strategy developed in [2].

Our experiments showed that: (i) For all protocols a small support size suffices for efficient communication in the extended setup. (ii) For almost all inputs considered the runners protocol outperforms the other two, with the hunters protocol having in almost all cases the worst performance both w.r.t. communication time and memory usage. Hence, it seems that this particular gnostic protocol cannot take advantage of global knowledge. (iii) There are no big differences in the relative performance of the protocols w.r.t. communication time and memory usage between the two motion patterns (random and adversarial) chosen by a mobile host. This suggests that random walks is a rather good pattern for modeling the motion of hosts. (iv) The throughput of all protocols scale almost linearly with the number of mobile hosts in the network. Hence, semi-compulsory, relay protocols are (perhaps) a better alternative to the path construction approach.

We tried to explain our experimental findings by unifying the protocols under a rather general and simple queuing-theoretic model (Section 3) which allows us to provide gross estimates for some of our measurements.

2 The Model of Motions

Throughout the paper with *mobile hosts* (or mobile users) we shall refer to those hosts (or users) that do not belong to the support Σ. Let u be the number of mobile hosts and let $|\Sigma| = k$.

We distinguish between two types of models: one type models the motion space, while the other type models the kind of motions the mobile hosts perform in the motion space.

2.1 The Motion Space

To model the motion space, we use the graph theoretic model introduced in [11] and also used in the previous studies [3, 4] that maps the movement of mobile hosts in the three-dimensional space S (with possible obstacles) to a so-called *motion graph* $G = (V, E)$ (which will be defined shortly). Let $|V| = n$. The environment where the stations move is abstracted by a graph by neglecting the detailed geometric characteristics of the motion. 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. To capture mobility of hosts in S, the space 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 $v \in V$ for each cube of volume $\mathcal{V}(tc)$. Two vertices v and w are connected by an edge $(v, w) \in E$ iff the corresponding cubes are adjacent. Let $\mathcal{V}(S)$ be the volume of space S. Clearly, $n = \mathcal{V}(S)/\mathcal{V}(tc)$ and consequently n is a rough approximation of the ratio $\mathcal{V}(S)/\mathcal{V}(tr)$.

For some protocols, a closely related graph is required. This graph is called the *cycle-graph* and was introduced in [2] for the pursuit-evasion game. A cycle-graph $G' = (V', E')$ of the motion graph G is a graph that has the same diameter D with G, consists of $\ell = O(n/D)$ disjoint cycles $C_1, \ldots, C_\ell$ of length D, has $|V'| = O(n)$, and $E' \supseteq E$. The cycle graph G' is constructed as follows. Start from a spanning tree of G, whose traversal yields a circuit of length $2n$. Partition the circuit into $\ell \leq 4n/D$ paths of length $\lceil D/2 \rceil$. Convert each path to a circuit by adding an edge (if it doesn't exist in G) between the first and the last vertex of the path. These circuits can be transformed into disjoint cycles as follows. Count, for each $v \in V$, the number $m(v)$ of times that it appears in some circuit. Create $m(v)$ copies of v in G'. For some vertex $v' \in V'$, let $or(v') \in V$ be the origin vertex of v' in V. For two vertices $w', v' \in V'$, there is an edge $(w', v') \in E'$, if either $(or(w'), or(v')) \in E$, or $or(w') = or(v')$. Clearly, G' consists of ℓ disjoint cycles, has the same diameter with G, and has in addition some extra edges.

2.2 The motion of the mobile hosts

We consider two kinds of motions that a mobile host can perform: random and adversarial.

Random motion is a natural starting point aiming at studying the average-case efficiency of protocols for basic communication. To this end, the motion of a mobile host is modeled by concurrent and independent random walks. This type of motion was used in [3, 4], assumes only local knowledge of G and hosts move using at most one edge in a single step. In fact, the assumption that the mobile hosts move randomly, either according to uniformly distributed changes in their directions and velocities, or according to the random waypoint mobility model by picking random destinations, has also been used in previous research (see e.g., [10, 13]).

Adversarial motion assumes that an oblivious adversary determines motion patterns in a way that a mobile host will meet a support member at the latest possible time. Hence, this kind of motion tries to identify the worst-case efficiency of a protocol. Such an adversarial motion pattern is adopted by the so-called rabbit strategy presented in [2]. This motion pattern assumes that the mobile hosts have complete knowledge of G and that are able to move at high speeds by traversing multiple edges of G in a single step (i.e., moving to non-adjacent vertices of G). The adversarial motion pattern is performed in rounds and is actually applied on the cycle-graph G' of G. The mobile host starts at a random position in some cycle C_i of G' and applies for $|C_i| - 1$ rounds a heavy-tailed random walk (for details see [2]) in a wrap around fashion on C_i. Then, it selects at random another cycle and repeats the procedure. Given an adversarial strategy of a mobile host h for G, a corresponding strategy in G' can be obtained by residing h at vertex v' of G', whenever h resides at vertex $or(v')$ of G.

In either the random or the adversarial case, we assume that the chosen motion pattern is performed *independently* of the motion of the support, i.e., we exclude the case where some of the mobile hosts are deliberately trying to avoid the members of the support during the whole execution of the protocol. This is a pragmatic assumption usually followed by application protocols.

3 The Relay Protocols

A semi-compulsory, relay protocol $\mathcal{P}$ uses the support Σ as an intermediate pool for collecting messages that are later delivered to their destinations. In other words, Σ plays the role of a (moving) backbone subnetwork through which all communication is routed. In this respect, $\mathcal{P}$ can be viewed as a queuing system (see Figure 1) having a three-phase routing policy.

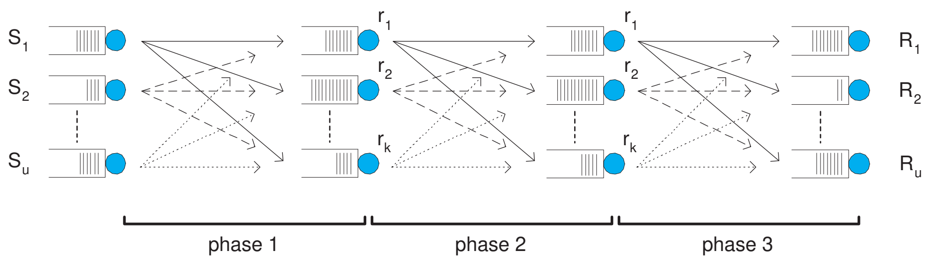

Figure 1: A Relay Protocol

In *Phase 1*, a message at a sending host i ($1 \leq i \leq u$) is served by the corresponding queue S_i and is forwarded to any of the support queues r_j ($1 \leq j \leq k$) – in fact, to the queue of the first support member j that comes within the transmission range of host i. In *Phase 2*, the packets received by the members of Σ are propagated within Σ. In *Phase 3*, the packets are delivered to their destination hosts R_l ($1 \leq l \leq u$).

The service rates at Phases 1 and 3 depend on the hitting (i.e., meeting) time of a mobile host (sender or receiver) and a member of the support. The service rate of Phase 2 depends on the particular protocol. For example, in the snake protocol the support packets are propagated sequentially within Σ and the service time is constant for each transmission between neighboring support members. In the runners protocol, the support packets are propagated within the support whenever two runners meet accidentally and thus the service time in this case depends on the meeting times of the independent walks of the runners.

A general way to model the above queuing system (ignoring Phase 2 which depends on the particular protocol) is as follows. Let λ_i be the *message generation rate* of mobile host i, i.e., the probability with which i generates new messages to be sent to various destinations. Let $\lambda^* = \sum_i \lambda_i$ be the total message generation rate in G. Each message m undergoes two sequential delays:

$D_1(m)$, being the delay in Phase 1 (how long m waits in the host it is generated till Σ gets it), and $D_3(m)$, being the delay in Phase 3 (how long m remains in Σ until it is delivered). Assuming that the members of Σ have unbounded buffer sizes and that λ^* is a Poisson stream, a simple model consists of two sequential *pure delay systems*, i.e., two systems in each of which there always exists an available (delay) server for each message. The probability distribution of the delay is allowed to be arbitrary. The distributions of D_1 and D_3 are the distributions of the mobile host hitting times with Σ. Let $\overline{S}_t$, $t = 1, 3$, denote the expected values of $D_t(m)$ (where the expectation is taken over the motion graph G, all mobile hosts, and all support motions). Then, the following can be easily proved (see [15, p.234,Ex.5.8]).

Lemma 1 *Let P_b be the steady state probability of b messages in the pure delay system t ($t = 1, 3$). Then,*

$$P_b = \frac{(\lambda^* \overline{S}_t)^b}{b!} e^{-\lambda^* \overline{S}_t}$$

regardless of the form of the distribution of $D_t(m)$.

The above lemma implies that the expected number of messages waiting in mobile hosts to be taken by the support is $\lambda^* \overline{S}_1$ (Phase 1) and that the expected number of messages waiting in the support until they delivered to the mobile hosts is $\lambda^* \overline{S}_3$ (Phase 3).

The above queuing-theoretic point of view allows us to select feasible values for the message generation rate λ_i of mobile host i. In the following, we shall assume that all hosts have the same message generation rate, i.e., $\lambda_i = \lambda$ for all hosts i. From a fundamental result of queuing theory, stating that for a system to be stable the input rate must be at most its service rate, we can obtain a gross estimation for λ. For example, by performing a few experiments on a random graph of size $n = 1600$ and support size $k = 10$, we observe that a message gets on the average about 33 rounds to be delivered. This is actually the service time of the queuing system, implying that $\lambda \leq 0.03$.

In the rest of this section, we shall briefly describe the protocols we considered in our study. In the following, we shall refer to the part of a protocol which determines the way in which the members of Σ move and communicate as the *support management module*, which consists of a *motion subprotocol*, a *sensor subprotocol* and a *synchronization subprotocol*.

3.1 The Snake Protocol

The main idea of the protocol proposed in [3] is as follows. There is a set-up phase of the ad-hoc network, during which a predefined number, k, of hosts become the members or nodes of the support. These hosts perform a leader election, which is run once and imposes only an initial communication cost. The elected leader, denoted by M_0, is used to co-ordinate the support topology and movement, and assigns local names to the rest of the support members M_1, M_2, ..., M_{k-1}.

The motion of the support members 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 M_0 doing a random walk in the motion graph – traversing only one edge per single step (or round) – and each of the other support members M_i executing the simple protocol "move where M_{i-1} was before". When some member 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 member of the support using a synchronization subprotocol P_3. When a receiver comes within the communication range of a member 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.

The analysis in [3] shows that the average message delay of the snake protocol, for the case where the hosts are performing random walks, is bounded above by the formula $\frac{2}{\lambda_2(G)}\Theta(n/k) + \Theta(k)$

where $\lambda_2(G)$ is the second eigenvalue of the motion graph G. More details for the snake protocol can be found in [3, 5]. In the latter paper, a more general case of motions is studied.

3.2 The Runners Protocol

A different approach to implement the support management module [4] 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 – traversing only one edge in a single step (or round). 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. However, all communication is still relayed through the support Σ. 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, the underlying sensor sub-protocol P_2 notifies the sender that it may send its message(s). When a mobile host comes within the communication range of a member of the support which has a message for a designated receiver $\mathcal{R}$, the waiting messages are forwarded to $\mathcal{R}$. The runners protocol is presented also in [6] in a more formal and complete way. In that paper mechanisms to reduce the memory requirements for the support's members are also discussed.

3.3 The Hunters Protocol

In this section we present yet another approach to implement the support management module, which is similar to the runners protocol in the sense that the members of Σ move independently from each other, but not in a fully random way. This approach is based on the hunter strategy presented in [2], and thus we will call it the *hunters protocol*. This strategy refers to the pursuit-evasion game on a graph. In this game there are two players, the hunter and the rabbit, who move along the vertices of the graph. The hunter is trying to catch a rabbit (i.e., is trying to move to the vertex on which the rabbit resides) and the rabbit is trying not to be caught (by moving to other vertices or stay idle). In our case the support members will play the role of the hunter, while the mobile hosts play the role of the rabbit.

Since the hunter strategy requires complete knowledge of the graph on which it is applied, the hunters protocol is a gnostic one. In particular, the following assumptions have to be made (in accordance with [2]): (i) each member of Σ has complete knowledge of the motion graph G; (ii) the members of Σ are able to occasionally move very fast, in the sense that they can traverse more than one edge of G in a single round of the protocol. Note, however, that while making such fast moves they are unable to transmit or receive any messages.

We will now proceed by describing the motion subprotocol P_1'' that dictates the way that the members of Σ move. During the set-up phase of the protocol, a set of k hosts become the members of Σ, and they elect a leader. The leader constructs the cycle graph G' and transmits it to the other members of Σ. Based on G', each member of Σ applies the hunter strategy for pursuit-evasion proposed in [2], which is the continuous repetition of the following three steps:

1. Select randomly a cycle C_i from $C_1, \ldots, C_\ell$.
2. Choose a random starting position at C_i (this takes at most $|C_i|/2$ rounds).
3. Move on C_i in a clockwise direction for the next $|C_i|$ rounds.

In each round the hunter is restricted to move using at most one edge of G'. Note that this may not be the case for his movement in G. Recall that there are some extra edges in G' which may not exist in G, and which in turn imply that the hunter is allowed to make jumps in G (i.e., to move to non-adjacent vertices by traversing consecutive edges in a single round).

It is proved in [2] that every hunter strategy on G' can be simulated on G without increasing the expected number of rounds to meet a mobile host. This simulation is carried out by simply residing the hunter at vertex $or(v')$ of G, whenever the hunter resides at vertex v' of G'. In [2] it is also shown that the adversarial motion pattern can cause a hunter to require $\Omega(n \log n)$ expected number of rounds to meet a single host (rabbit) and that the above hunter strategy actually achieves an $O(n \log n)$ expected number of rounds.

Finally, in a similar way to the runners protocol, the hunters will exchange messages whenever they meet accidentally on the same vertex of G by using the synchronization subprotocol P_3'.

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 [18]. We used the simulation environment developed in [3, 4] (*adhocsim*) where each class is installed in an environment that allows to read graphs from files as well as to perform a network simulation for a given number of rounds, a fixed number of mobile hosts 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.

In the previous experimental works [3, 4] we evaluated the average-case performance based on the assumption that the hosts are performing random walks on the motion graph G. Since the hosts had no global knowledge of G, the movement of mobile hosts was provided by the so-called `environment` class which simply selected a random vertex adjacent to the current position of the host. For purposes of studying the efficiency of the algorithms in the worst-case (using the adversarial strategy) and implement the Hunters protocol, we had to redesign our previous implementation so that the mobile hosts are able to make decisions on their next position inside the `mobile_host` class since this requires knowledge of the motion graph G. In particular, based on the object behavioural patterns presented in [7], we defined a family of motion strategies, each of which inherits from a base-class (`moveStrategy`) and encapsulates a different motion strategy. In this way, these motion strategies can be used interchangeably by the mobile hosts. The `moveStrategy` class declares an interface common to all supported algorithms that are based on the `mobile_host` class. We then extended the `moveStrategy` class in order to implement the classic random walk (`moveRW`) that is used by the runners and the head of the snake, the follower strategy (`moveFollower`) that is used by the tail of the snake, the hunters strategy (`moveHunter`) and finally the adversarial strategy (`moveRabbit`).

4.2 Experimental Setup

A large number of experiments were carried out modeling the different possible situations regarding the geographic area covered by an ad-hoc mobile network and the motion of hosts within it. Each experiment is characterized by a tuple of parameters $\langle \mathcal{P}, k, u, \mu, \lambda, G, n \rangle$, where $\mathcal{P}$ is the protocol used, $k = |\Sigma|$, u is the total number of mobile hosts h ($h \notin \Sigma$), μ is their pattern of movement, λ is the message generation rate by each host per round, $G = (V, E)$ is the motion graph, and $n = |V|$. Note that in this work, we are not interested in examining the behaviour of protocols in the presence of failures of hosts or message losses due to communication failures.

We considered four kinds of motion graphs (described below), several values for n in the range $[400, 6400]$, and different values of k in $[3, 50]$.

Each experiment 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 and resides in a particular vertex v_h of the motion graph G. Two mobile hosts i, j communicate by sending messages if they reside on the same vertex of the motion graph (i.e., $v_i = v_j$). We assume that in this case message exchange takes negligible time (i.e., the messages are short packets and the wireless transmission is very fast).

In a simulation round each mobile host h moves to a vertex of the motion graph G, and performs some computation. We assume that a mobile host can receive several messages and send several messages at the same round. If $h \in \Sigma$, its move and local computation are determined by the protocols described in Section 3. If $h \notin \Sigma$, then its move is determined by the strategy μ and with probability λ the host h generates a new message (e.g., if $\lambda = 0.01$ a new message is generated roughly every 100 rounds) by picking a random destination host.

For each motion graph G considered, we used various numbers of mobile hosts u, namely $u \in [500, 5000]$, which were injected at random positions of G and generated message exchanges with $\lambda \in [0.001,\ 0.025]$. The selection of this value for λ is based on the assumption that the mobile hosts do not execute a real-time application that requires continuous exchange of messages, but can tolerate delays in the order of fractions of seconds to several seconds. Our focus is on applications that are so asynchronous in nature that they can tolerate end-to-end delays. Examples of such applications include electronic mail, database synchronization between a mobile terminal and a central database, and certain types of event notification (similar assumptions are made in [8]).

Finally, regarding motion graphs, we considered the families introduced in [3, 4] that model different geographic areas. These 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 vertices 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 $n \in \{1600, 3200, 6400\}$.

Grid Graphs. We consider 2D and 3D grid graphs. The former 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 $n \in \{400, 1600\}$. The latter helps in evaluating the performance of the protocols when the mobile hosts move in the 3D space. We considered three different 3D grid graphs ($n^{1/3} \times n^{1/3} \times n^{1/3}$) with $n \in \{512, 1000, 1280\}$.

Bipartite multi-stage graphs. A bipartite multi-stage graph is a graph consisting of a number of stages 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 3, 7, 8 or 9 stages, and number of vertices $n \in$\{1600, 3200, 6400\}.

Two-level motion graphs. Motivated by the fact that most mobile hosts 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.

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 hosts of the network (i.e., the sender-receiver pairs are not fixed).

4.3 Experimental Results and Discussion

We carried out each experiment for a sufficiently large number of rounds (at least $7,000$) during which a sufficiently large number of messages (at least $100,000$) were delivered to the designated receivers. We used the following metrics to evaluate the performance of each protocol: average message delay, memory usage, and throughput.

Average Message Delay. We started our experimentation by measuring the average message delay for the three protocols over different support sizes k. In particular, we measured the total delay of messages exchanged between pairs of sender-receiver mobile users. For each message generated, we calculated the overall delay (in terms of simulation rounds) until the message was finally transmitted to the receiver.

The reported experiments for the four different motion graphs we considered are illustrated in Figures 2, 3, 4, 5, and 6. In these graphics, we have included the results for motion graphs of size $n = 1600$, $u = 1000$, and $\lambda = 0.01$ (similar results hold for other values of n, u and λ). The first three figures show the measured average message delays in the case where the mobile users are performing concurrent random works ($\mu =$random), while the last two are for the case where the mobile users are using an adversarial strategy ($\mu =$adversarial). The curves reported for the snake and runners protocol match the ones reported in [3, 4]. 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.

Regarding the hunters protocol, we firstly observe that its curve follows the same pattern with that of the other two protocols. Unfortunately, it is not easy to extend the theoretical analysis in [2] in order to study the effect of the value of k on the average message delay, but from the experimental evidence we suspect that it obeys a similar bound with that of the other two protocols. Our second observation is that the performance of the hunters protocol seems to be less correlated than the snake and runners protocol with the type of the motion graph. To explain this, we first note that the grid graphs have a smaller second eigenvalue than that of bipartite multi-stage and random graphs. For the snake protocol, a small value for this eigenvalue makes the average message delay to be more dependent on the hidden constant in the asymptotic analysis [3] (see also Section 3.1). Although this is apparently larger than the corresponding constant of the runners protocol, it seems that the constant in the hunters protocol is smaller. This could be explained by observing the curves of Figures 2, 3, 4. For the case of the two level motion graphs, the performance of the hunters protocol is better than that of the snake. Moving on to the case of 3D grid-graphs (i.e., for graphs of larger second eigenvalue), the performance of the snake and hunters protocol is almost the same. Finally, for the case of bipartite multi-stage graphs, the performance of the hunters protocol degrades further making it less efficient than the snake protocol.

In the case where the mobile hosts use an adversarial strategy (see Fig. 5 and 6), there is a similar (to the random case) threshold behaviour w.r.t. k. However, we interestingly observe that both the snake and runners protocols outperform the hunters protocol for all cases considered. For the case of 3D grid-graphs (see Fig. 5), it seems that the performance of the hunters protocol is almost two times worse than that of the snake protocol, while for the bipartite multi-stage (see Fig. 6) the gap increases to five times (a similar gap applies also in the case of random graphs). This can be partially justified by the fact that in [2], the $O(n \log n)$ expected performance of the hunter strategy is investigated for the case of one hunter against one rabbit. In our experiments we have considered multiple hunters that try to catch a large number of rabbits which deviates from the model studied in [2].

A different possible explanation of this observation, is that the snake and runners protocols are designed to perform random walks as a result of which they manage to meet a large number of mobile hosts within the cover time of the motion graph (the expected time to visit all vertices). In contrast to this, the hunters, in their effort to catch a single mobile host, they make jumps between distant nodes of the graph. Although this improves their chances to catch this host, they apparently "miss" a number of mobile hosts that happens to reside on intermediate nodes of G.

Memory Usage. In the second set of experiments we were interested in examining the number of messages waiting in Σ to be delivered at any given time. We executed a number of experiments, for different types and sizes of motion graphs. During each simulation round we measured the number of messages stored (buffer size) in each member of Σ. Based on these measurements, we obtained the average number of messages stored in a support member and the maximum number of messages ever stored in a member of Σ (among all members).

We report results on a fixed number of mobile hosts $u = 1000$ (similar results hold for other values of u) and various message generation rates (λ). We start with the case of random motion pattern for the mobile hosts. Figures 7 and 8 show the average buffer size and the maximum buffer size per member of the support for a random graph of size $n = 6400$ and a support size $k = 50$ (similar results hold for other types and sizes of motion graphs, and values of k).

The reported curves of Fig. 7 clearly show that the average number of messages that are stored in Σ and are waiting to be delivered grows almost linearly with the message generation rate λ. Furthermore, we observe, in conjuction with Fig. 4, that the buffer sizes (of the members of Σ) are also affected by the average message delay. The average and maximum buffer sizes for the snake and runners protocol are almost identical since the corresponding average message delays are similar (see Fig. 4). For the hunters protocol, the buffer sizes grow faster as the message generation rate increases, since the average message delay is higher than that reported for the snake and runners protocol (see Fig. 4). This observation confirms the theoretical results of Lemma 1 which implies that the expected number of messages waiting in Σ until they get delivered to the mobile hosts is $\lambda\overline{S_3}$, where $\overline{S_3}$ is the average hitting (i.e., meeting) time of a mobile host and a member of Σ.

In order to provide estimates on the buffer size (i.e., memory) requirements of each member of Σ, we can consult Figure 8. In fact, the implementor of the protocol, based on the actual needs of the hosts (i.e., the message generation rate), the area covered by the mobile network (i.e., the type and size of the motion graph), and the available number of mobile hosts that will form the support, can estimate the requirements of the support members in terms of memory capabilities. Alternatively, if the memory requirements of the members of Σ are apriori fixed, then by fine tuning k (and thus the expected average message delay) can provide a guarantee for the performance of the network (in terms of rejected and/or lost messages).

Regarding adversarial motion of hosts, the experimental results we obtained are similar to the random motion case, though worse w.r.t. the buffer size. This is not a surprise, since $\overline{S_3}$ is larger in this case.

Throughput. In the third set of experiments we opted at evaluating the overall throughput of the three protocols studied. Based on the model presented in Section 3, we are interested in experimentally evaluating the total number of messages delivered (Phase 3) by the system over a given period of time (rounds). Let us first attempt to provide a theoretical estimate on the throughput.

Let $M(t)$ denote the number of messages delivered in t rounds. The steady state expected throughput x (number of delivered messages per time unit) of the system is

$$x = \lim_{\Delta T \to \infty} \frac{M(\Delta T)}{\Delta T}$$

Now, let C be a random variable denoting the time to visit all vertices of the motion graph G; thus, $E(C)$ is the *cover time* of G (expected time to visit all vertices). Consider two consecutive time periods C_1 and C_2 of duration $|C_1|$ and $|C_2|$, respectively, during each of which the support visits all vertices. Let also assume that no message is generated by a host during C_2. Then clearly, any message generated during C_1 is delivered by at most the end of C_2. Let $\Delta T = |C_1| + |C_2|$. Then,

$$x = \frac{M(|C_1| + |C_2|)}{|C_1| + |C_2|} = \frac{\lambda^* |C_1|}{|C_1| + |C_2|}$$

To obtain a crude estimate on x, let us replace C_i by $E(C_i)$, $i = 1, 2$. Then, $E(C_1) = E(C_2)$ and consequently $x = \lambda^*/2$. That is, the throughput grows linearly with the total message generation rate.

The above rough theoretical estimation was fully confirmed by our experiments. We considered several types and sizes of motion graphs as well as message generation rates and total number of mobile hosts. In particular, for a period of (at least) 7000 rounds we measured the total number of messages delivered over all sender-receiver pairs.

Figures 9 and 10 illustrate the total throughput of the three protocols for random motion of mobile hosts. (Similar results hold for adversarial motion, since the throughput is independent of the motion of mobile hosts.) Interestingly, all protocols achieve almost the same throughput which turns out to be linear both to the message generation rate (Fig. 9) and to the number of mobile users (Fig. 10). This comes directly in contrast to the fundamental results of [9] that study the throughput (or capacity) of wireless networks using the path construction approach. In fact, our experimental outcomes show the superiority of the relay protocols, regardless of the motion strategy implemented.

5 Closing Remarks

We have presented the implementation of a new protocol (hunters) for basic communication in ad-hoc mobile networks that requires global knowledge of the motion space and performed a comparative experimental study with other two protocols (snake, runners) that they require only local knowledge. All protocols belong to the semi-compulsory, relay category. Our experiments showed that hunters are almost always inferior to the other two. The throughput of all protocols, however, scale linearly with the number of mobile hosts in the network, demonstrating the superiority of semi-compulsory, relay protocols against other approaches.

References

[1] M. Adler and C. Scheideler: Efficient Communication Strategies for Ad-Hoc Wireless Networks. *Theory of Computing Systems*, 33(5/6):337-392, 2000.

[2] M. Adler, H. Räcke, N. Sivadasan, C. Sohler, and B. Vöcking. Randomized Pursuit-Evasion in Graphs. To appear in *Proc. 29th Int'l Conf. on Automata, Languages, and Programming* – ICALP'02, 2002.

[3] 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.

[4] I. Chatzigiannakis, S. Nikoletseas, N. Paspalis, P. Spirakis and C. Zaroliagis: An Experimental Study of Basic Communication Protocols in Ad-hoc Mobile Networks. In *Proc. 5th Annual Workshop on Algorithmic Engineering* – WAE'01, 2001.

[5] I. Chatzigiannakis, S. Nikoletseas, and P. Spirakis: An Efficient Communication Strategy for Ad-hoc Mobile Networks In *Proc. 15th International Symposium on Distributed Computing* – DISC'01, 2001. Also Brief Announcement in *20th Annual Symposium on Principles of Distributed Computing* – PODC'01, 2001.

[6] 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 IPDPS'02, 2002.

[7] E. Gamma, R. Helm, R. Johnson and J. Vlissides. *Design Patterns.* Addison-Wesley Professional Computing Series, 1995.

[8] M. Grossglauser and D. Tse: Mobility Increases the Capacity of Ad-hoc Wireless Networks. In *Proc. 12th Annual IEEE Conference on Computer Communication* – INFOCOM'01, 2001.

[9] P. Gupta, P. R. Kumar: The Capacity of Wireless Networks. IEEE Transactions on Information Theory, 46(2):388-404, March 2000 (2000).

[10] Z. J. Haas and M. R. Pearlman: The performance of a new routing protocol for the reconfigurable wireless networks. In *Proc. ICC'98*, 1998, pp.562-566.

[11] 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.

[12] R. W. Hogg, A. L. Rankin, M. C. McHenry, D. M. Helmick, C. F. Bergh, S. I. Roumeliotis, L. Matthies: Sensors and Algorithms for Small Robot Leader/Follower Behavior. In *Proc. 15th Annual Symposium on AeroSense* – SPIE'01, 2001.

[13] 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.

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

[15] L. Kleinrock, *Queuing Systems, Theory,* Vol I, John Wiley Editions, 1975.

[16] J. Li, C. Blake, D.S.J. De Couto, H.I. Lee and R. Morris: Capacity of Ad-hoc Wireless Networks. In *Proc. 7th Annual ACM/IEEE International Conference on Mobile Computing* – MOBICOM'01, 2001.

[17] 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.

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

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

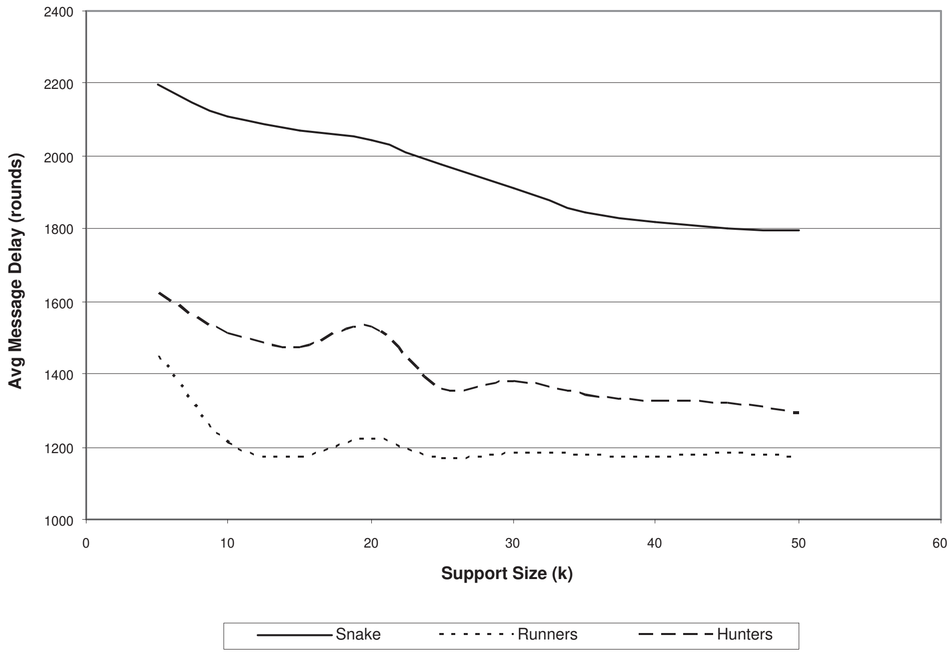

Figure 2: Average Message Delay for two-level graphs ($n = 1600$, μ =random).

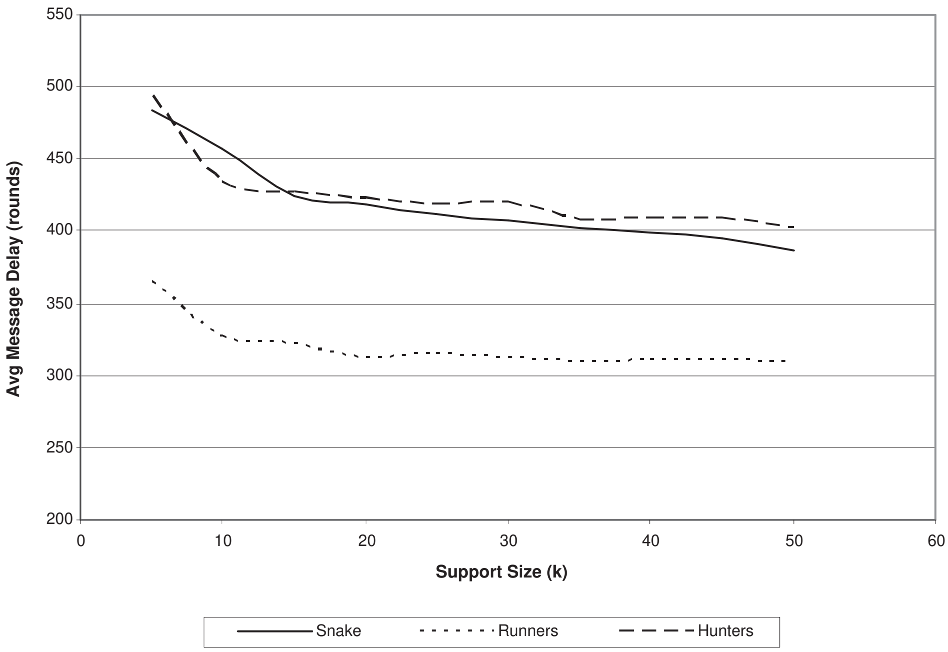

Figure 3: Average Message Delay for 3D grid graphs ($n = 1728$, μ =random).

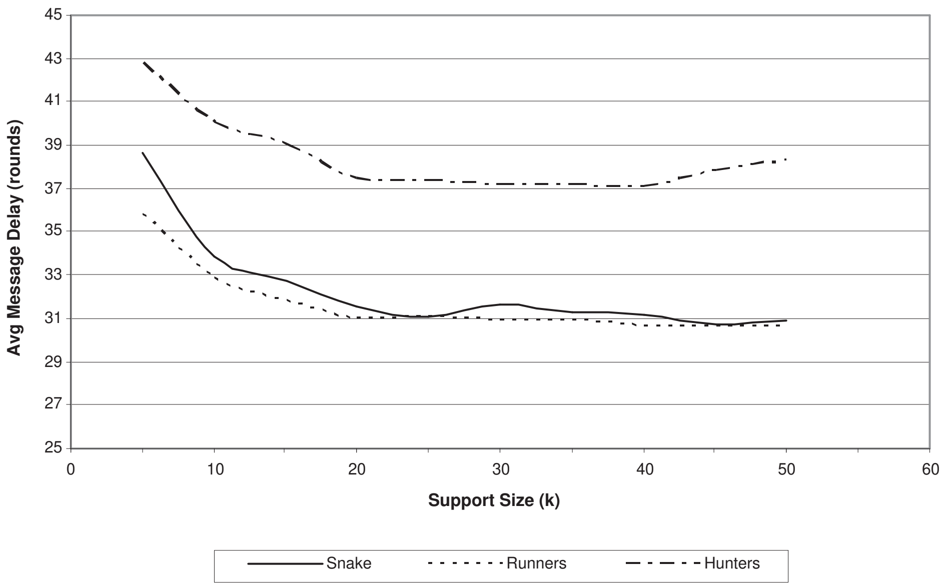

Figure 4: Average Message Delay for random graphs ($n = 1600$, μ =random).

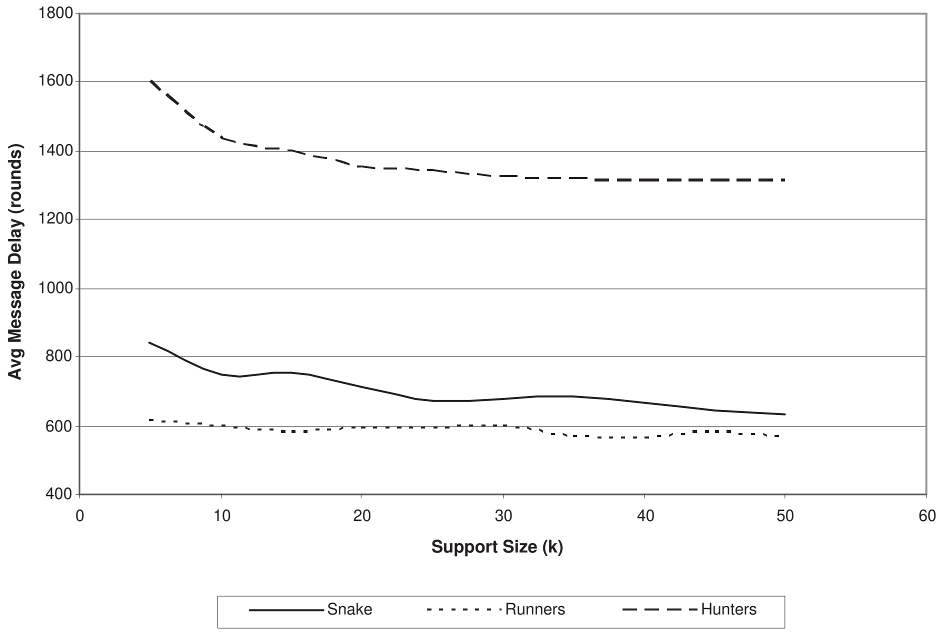

Figure 5: Average Message Delay for 3D grid graphs ($n = 1728$, μ =adversarial).

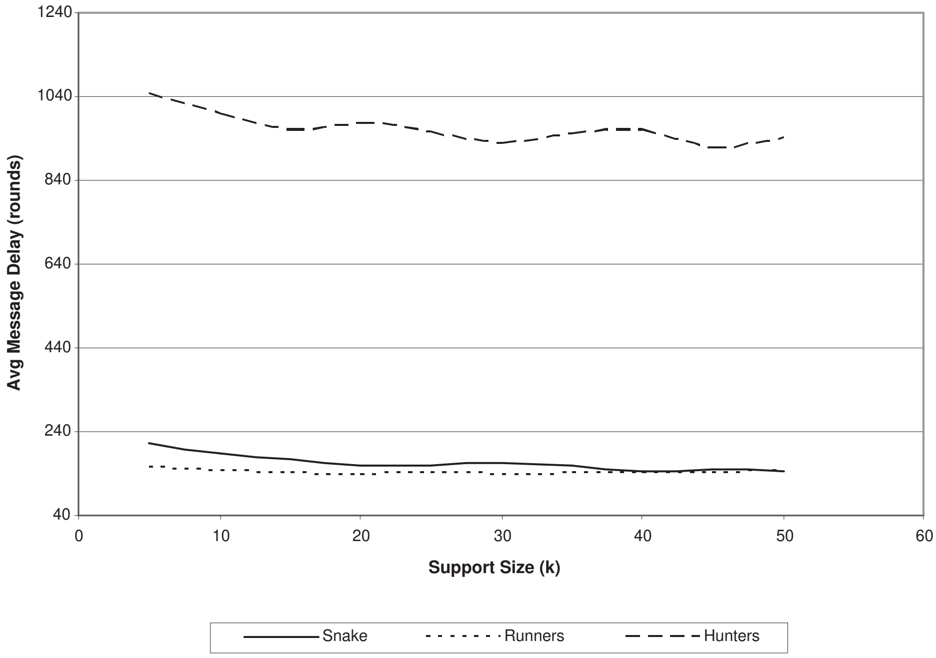

Figure 6: Average Message Delay for bipartite multi-stage graphs ($n = 1600$, $\xi = 3$, μ =adversarial).

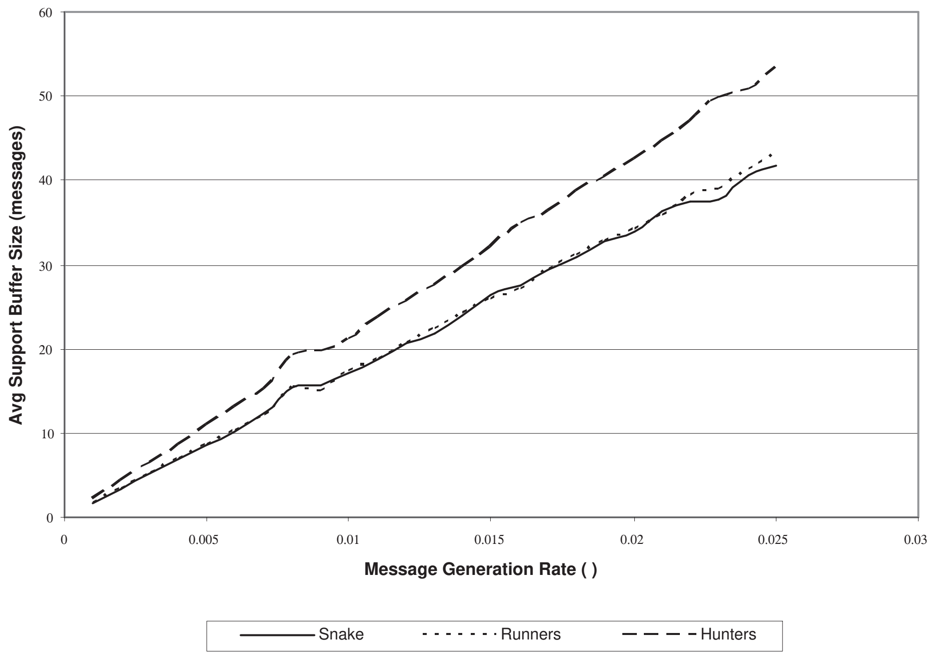

Figure 7: Average Buffer Size (messages) per Support Member for random graphs ($n = 6400$, $u = 1000$).

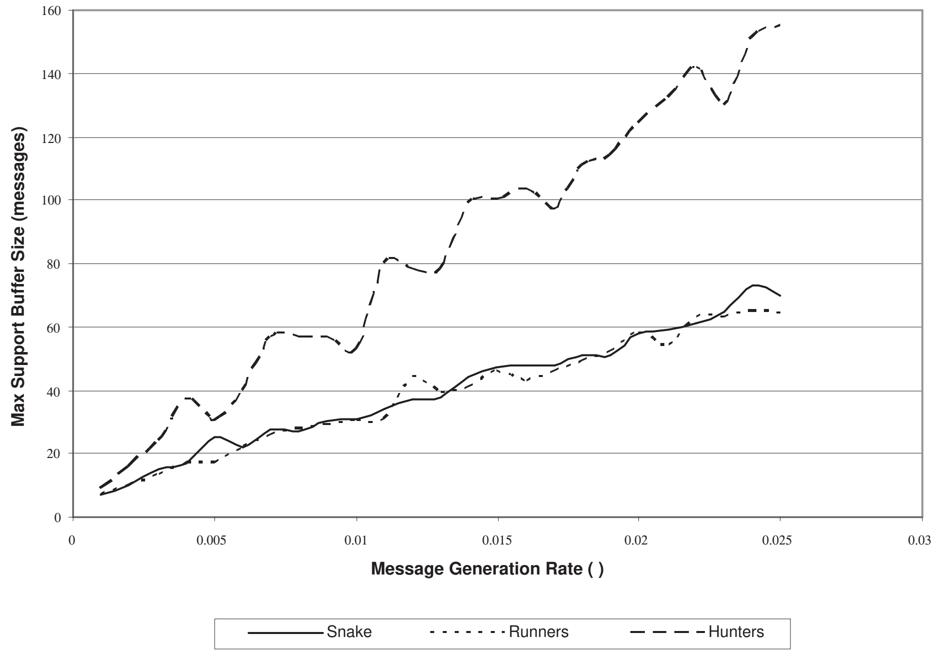

Figure 8: Maximum Buffer Size (messages) per Support Member for random graphs ($n = 6400$, $u = 1000$).

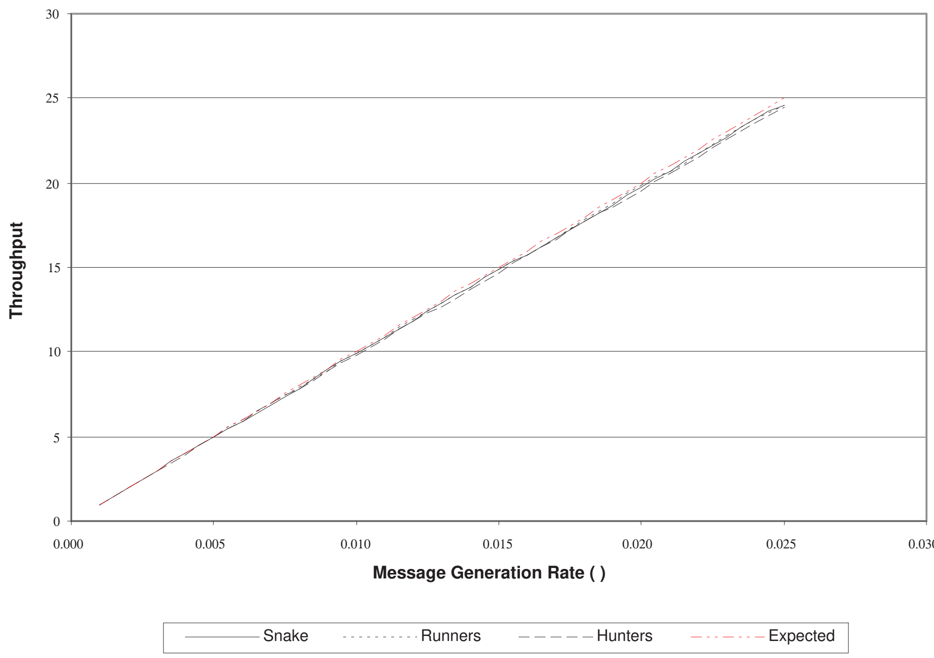

Figure 9: Maximum Throughput for random graphs ($n = 6400$), $u = 1000$ and various message generation rates λ.

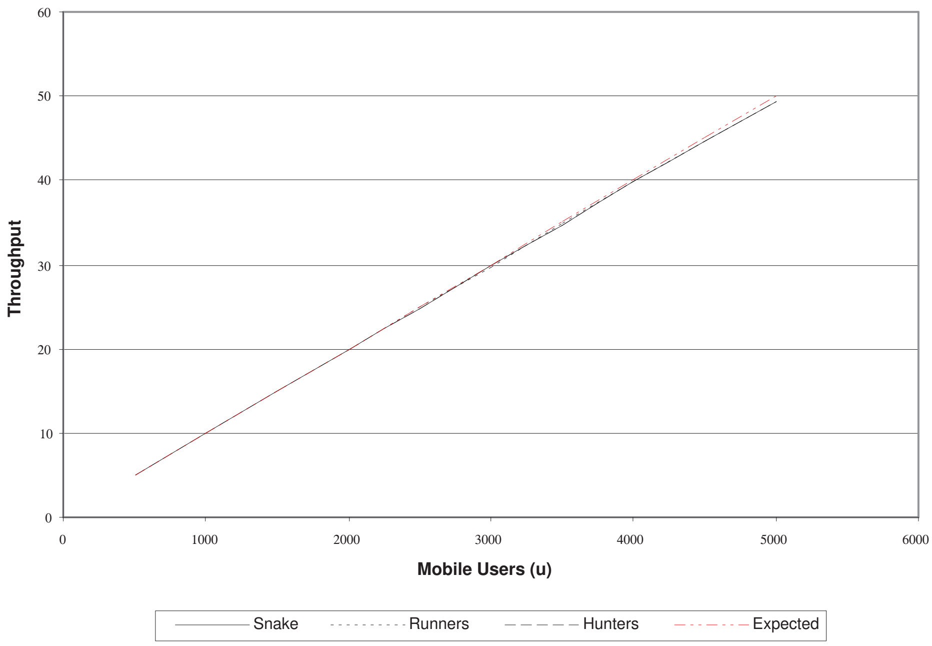

Figure 10: Maximum Throughput for random graphs ($n = 6400$), $\lambda = 0.01$, and various number of mobile hosts u.