Virtual Mobile Nodes for Mobile *Ad Hoc* Networks

(Extended Abstract)

Shlomi Dolev* Seth Gilbert† Nancy A. Lynch† Elad Schiller*

Alex A. Shvartsman‡† Jennifer Welch§

February 24, 2004

Abstract

One of the most significant challenges introduced by mobile networks is the difficulty in coping with the *unpredictable* movement of mobile nodes. If, instead, the mobile nodes could be programmed to travel through the world in a predictable and useful manner, the task of designing algorithms for mobile networks would be significantly simplified. Alas, users of mobile devices in the real world are not amenable to following instructions as to where their devices may travel.

While real mobile nodes may be disinclined to move as desired, we propose executing algorithms on *virtual mobile nodes* that move in a predetermined, predictable, manner through the real world. In this paper, we define the Virtual Mobile Node Abstraction, and present selected algorithms that take advantage of virtual mobile nodes to simply and efficiently perform complicated tasks in highly dynamic, unpredictable mobile *ad hoc* networks.

We then present the *Mobile Point Emulator*, a new algorithm that implements robust virtual mobile nodes. This algorithm replicates the virtual node at a constantly changing set of real nodes, choosing new replicas as the real nodes move in and out of the path of the virtual node. We claim that the Mobile Point algorithm correctly implements a virtual mobile node, and that it is robust as long as the virtual node travels through well-populated areas of the network.

Contact Author: Seth Gilbert, 617-283-7502

Contact Address: 70 Pacific Street #666B, Cambridge, MA. 02139

Keywords: mobile networks, *ad hoc* networks, highly dynamic distributed algorithms, fault-tolerance, location-aware

Pages: 10 page main paper, 2 page bibliography, 4 page appendix

Students: Seth Gilbert and Elad Schiller are students.

Brief Announcement: Consider for Brief Announcement *if* not accepted as a regular presentation.

Program Committee: Alex A. Shvartsman is on the program committee.

*Ben-Gurion University, {dolev,schiller}@cs.bgu.ac.il

†MIT CSAIL, {sethg,lynch,alex}@theory.lcs.mit.edu

‡Department of Computer Science and Engineering, University of Connecticut, aas@cse.uconn.edu

§Texas A&M University, welch@cs.tamu.edu

[1]This work is supported in part by NSF grant CCR-0098305 and NSF ITR Grant 0121277. Part of the work of the first author and four author has been done during visits to MIT and Texas A&M. The first author and fourth author are partially supported by an IBM faculty award, the Israeli ministry of defense, NSF, and the Israeli Ministry of Trade and Industry. The second and third authors are partially supported by AFOSR Contract #F49620-00-1-0097, DARPA Contract #F33615-01-C-1896, NSF Grant 64961-CS, NTT Grant MIT9904-12. The fifth author is partially supported by the NSF Grant 9988304, 0311368 and by the NSF CAREER Award 9984774. The sixth author is partially supported by NSF Grant 0098305 and Texas Advanced Research Program 000512-0091-2001.

1 Introduction

Devising algorithms for highly dynamic mobile networks is hard. The resulting algorithms have complicated specifications, are difficult to understand, and challenging to implement. It is therefore important to develop techniques and abstractions that simplify the process of developing and implementing algorithms for mobile networks. In this paper, we propose one such abstraction, the Virtual Mobile Node model.

Compounding the challenges of designing algorithms for typical dynamic distributed systems, highly mobile networks introduce the difficulty of dealing with the completely *unpredictable motion* of the nodes. This complication is of course unavoidable: the defining feature of a mobile network is that the nodes do, in fact, move. Another difficulty in dynamic settings is the *unpredictable availability* of nodes that continually join and leave the system, whether they do so voluntarily, or are turned on and off, or fail and recover.

The task of designing algorithms would be significantly simplified if the mobile nodes were reliable and moved in predictable ways. The mobility of the nodes, however, also presents an opportunity: if the mobile nodes were to move in a *useful* way, it would be possible to take advantage of the motion to design algorithms that are even more efficient than those for static networks. This idea is illustrated by Hatzis et al. in [11], which defines the notion of a *compulsory* protocol, one that requires a subset of the mobile nodes to move in a specific manner. (In particular, in [11], the nodes are required to perform a random walk.) They present an efficient compulsory protocol for leader election, demonstrating the advantages over a non-compulsory protocol. The routing protocols of Chatzigiannakis et al. [6] and Li et al. [18] provide further evidence that compulsory protocols are simple and efficient.

Compulsory protocols deploy dedicated nodes that are committed to certain patterns of motion, and this may be a very strong requirement for highly dynamic asynchronous mobile settings where nodes may fail or be diverted from the prescribed path. Thus the our objective is (a) to retain the effectiveness of the compulsory protocols, and (b) achieve this without imposing requirements on the motion of the nodes. In this paper we introduce an approach to building distributed applications that achieves both objectives.

Our Contributions. In this paper we introduce the Virtual Mobile Node Abstraction, and show how it can be used to build distributed applications for dynamic *ad hoc* networks, thus demonstrating the utility of our approach. We also demonstrate the feasibility of our approach by showing how to implement the Virtual Mobile Node Abstraction.

Virtual Mobile Nodes. We propose designing distributed algorithms to run on *virtual mobile nodes* (VMNs), abstract nodes that move in a predetermined, predictable manner. In this new abstraction, VMNs are simulated by real nodes in the network. The motion of a VMN is determined in advance, and is known to the programs executing on the VMNs. For example, a VMN may traverse the plane in a regular pattern, or it may perform a pseudorandom walk.

To be effective, we allow the motion of the virtual nodes to be uncorrelated with the motion of the real nodes: even if all the real nodes are going in one direction, the virtual nodes are able to travel in the opposite direction. Consider, for example, an application to monitor traffic on a highway: even though all the cars are moving in one direction, a VMN could move in the opposite direction, notifying oncoming cars of the traffic ahead.

Algorithms for Virtual Mobile Nodes. To demonstrate the utility of the new approach, we present selected algorithms that use VMNs to solve interesting problems simply and efficiently. We first consider the problem of routing messages. We describe two schemes for reliably delivering messages: one relies on the compulsory protocols of Chatzigiannakis et al. [6] to route messages among the real nodes; the other delivers messages only to virtual nodes. The second scenario explores the problem of collecting and evaluating real-time data in mobile *ad hoc* sensor networks. In the third, we consider a number of general services, such as group communication services (e.g., as in [10, 14, 15]) and an atomic memory service. We solve these problems using approaches developed by Dolev et al. [9] and Dolev et al. [7].

Implementing Virtual Mobile Nodes. We present the *Mobile Point* algorithm, a new algorithm that implements robust virtual mobile nodes, thus demonstrating the feasibility of our approach. The main idea of the algorithm is to allow real nodes traveling near the location of a virtual node to emulate that VMN. In order to achieve robustness, the algorithm replicates the state of a virtual node at a number of real mobile nodes. As the execution proceeds, the algorithm continually modifies the set of replicas for each mobile node so that the replicas always remain near the desired path of the virtual node. We use a replicated state machine approach, augmented to support joins, leaves, and recovery, to maintain the consistency of the replicas.

A virtual node is prone to "crash-reboot" failures. As long as the path of the virtual node travels through dense areas of the network, the virtual node does not fail. If however, the VMN is directed to an empty spot, a failure may occur. The Mobile Point algorithm, however, allows the VMN to recover to an initial state, if it again reenters a dense area. In this way, the VMN abstraction takes advantage of dense regions of the network to perform arbitrary computation.

To summarize, this paper contains three main contributions. First, we define the VMN abstraction. Second, we present selected algorithms based on VMNs that are quite simple compared to previous algorithms. Third, we present an algorithm to implement robust virtual mobile nodes.

Other Related Work. The Mobile Point algorithm for implementing VMNs is similar in many ways to the GeoQuorums algorithm [7, 8]. Dolev et al. define a Focal Point abstraction that allows mobile nodes to use predetermined locations in the network to simulate atomic objects. The Virtual Mobile Node Abstraction extends the Focal Point abstraction in four ways. First, in the GeoQuorums algorithm, the virtual entities (i.e., the focal points) are static: they are limited to fixed, predetermined locations. In this paper, we extend the abstraction technique to allow each virtual entity to move on an arbitrary, predetermined path. Second, the Focal Point abstraction limits the virtual entities to be atomic objects. In this paper, the virtual entities can emulate any arbitrary automaton. Third, the GeoQuorums algorithm does not allow the virtual entities to recover, should they fail, whereas we support recovery. Fourth, the GeoQuorums algorithm inherently depends on a GeoCast service that allows potentially more expensive long-distance communication. In this paper, we can implement the Virtual Mobile Node Abstraction even when the available communication services only allow local communication.

This paper also generalizes some results presented by Beal [2, 3], in which he defines a *PersistentNode* abstraction. A PersistentNode is a virtual entity that travels around a static (rather than mobile) sensor network. It can carry with it some state, but the consistency guarantees are relatively weak.

The work of Nath and Niculescu [21] also takes advantage of precalculated paths to forward messages in dense networks. Messages are routed along trajectories, where nodes on the path forward the messages. Similarly, prior GeoCast work (for example, [22, 5]) attempts to route data geographically. In many ways, these strategies are ad hoc attempts to emulate some kind of traveling node. We provide a more general framework to take advantage of dense areas of the network to perform arbitrary computation. A significant focus of these prior papers is *determining* good trajectories, a problem that we do not address (but that may be useful in determining a good path for VMNs).

Document Structure. In the first part of the paper, we show how to use virtual mobile nodes to simplify the design of algorithms for mobile *ad hoc* networks. We describe the underlying system model in Section 2, and the Virtual Mobile Node model in Section 3. Then, in Section 4, we describe selected algorithms that take advantage of the VMN abstraction. In the second part of the paper, we show how to implement robust virtual mobile nodes, using the Mobile Point algorithm in Section 5. Selected algorithms are given in the appendix.

2 Basic System Model

In this section, we present the underlying system model which consists of physical mobile nodes moving in a bounded region of a two-dimensional plane. Each mobile node is assigned a unique identifier from a set, I. The mobile nodes may join and leave the system, and may fail at any time. (We treat leaves as failures.) The real mobile nodes can move on any continuous path in the plane, with speed bounded by a constant, v_{max}.

The *Geosensor* is a component of the environment that maintains the current location of each mobile node. It also maintains the current real time. The Geosensor can be implemented in real systems by a Global Positioning System (GPS) receiver, or (for indoor usage), a Cricket [24] device.

Basic Communication Services. The mobile nodes support a basic broadcast communication service, LocalCast, which is parameterized by a radius, R. The R-LocalCast service delivers a message to every mobile node within a radius R of the sender. The service has the following properties: (i) *Reliable Delivery*: Assume that the mobile node i performs a $\mathsf{localcast}(m)_i$ action. Then for every mobile node j that is within distance R of the location of i when the message is sent, and remains within distance R of that location forever thereafter[1] and does not fail, a $\mathsf{localcast\text{-}rcv}(m)_j$ event eventually occurs, delivering the message to node j. (ii) *Integrity*: For any LocalCast message m and mobile node i, if a $\mathsf{localcast\text{-}rcv}(m)_i$ event occurs, then a $\mathsf{localcast}(m)_\ell$ event precedes it, for some mobile node ℓ. In Section 5, we assume that every message, that is delivered, is delivered within d time. We assume d is fixed for the rest of the paper. This service can be implemented in real systems as long as R is not too large. Many typical wireless broadcast protocols, augmented with error correction, should satisfy these requirements. In Section 6 we briefly discuss some weaker alternatives that can tolerate occasionally delayed and lost messages; these alternatives are even more realistic models of real systems.

3 Virtual Mobile Nodes

The *VMN Abstraction* consists of both physical mobile nodes (PMNs, also referred to as "real nodes") and virtual mobile nodes (VMNs, also referred to as "virtual nodes"). Throughout this paper, the term *mobile node* refers to any node in the system, be it real or virtual. Mobile nodes may fail and recover; when a node recovers, it begins again in an initial state.
The virtual nodes are equivalent to the real nodes, with a few exceptions. Most importantly, they move in a predictable, predetermined path, rather than an arbitrary path.

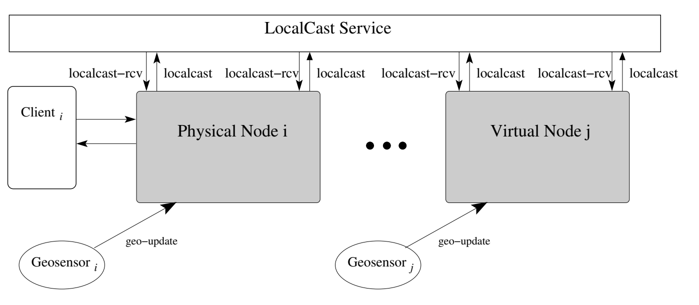

Figure 1: VMN System Model, representing a virtual node and a real node sending messages to each other via LocalCast service.

Each virtual node is designed to execute a regular I/O automaton[2] [19]. The VMN System Model supports the LocalCast service, with the same basic specifications as the underlying system model. In this case, the broadcast service does not distinguish between virtual and real nodes, delivering messages to all mobile nodes within the specified vicinity. The LocalCast supported by the VMN model, however, may delay messages from being sent or received, and may repeat and reorder messages (even if the underlying LocalCast service does not).

[1]As soon as the message is delivered, the receiver can move away. If the broadcast latency is bounded, this is effectively a bound on the speed of the mobile nodes.

[2]We expect that it is a simple extension to support timed and hybrid virtual nodes, instead of just regular I/O automata.

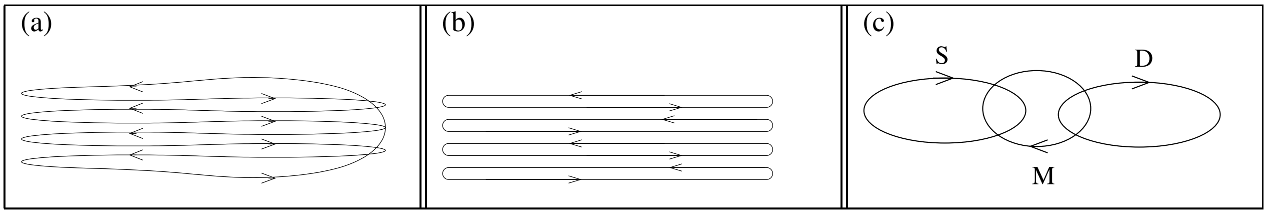

Figure 2: The diagrams represent one or more VMNs traveling in the plane. The arrows indicate the direction of motion. (a) A single virtual mobile node scans an area. (b) Multiple virtual mobile nodes cooperate to scan an area. Neighboring VMNs move counterclockwise, meeting once per cycle to exchange messages. (c) Messenger VMN M ferries messages from source VMN S to destination VMN D.

Since the automaton being executed is running on a (virtual) mobile node, we allow it to communicate only by sending and receiving messages, and by receiving updates from the Geosensor. That is, we limit the input and output actions of virtual node h to $\mathsf{localcast\text{-}rcv}(msg)_h$, $\mathsf{localcast}(msg)_h$, and $\mathsf{geo\text{-}update}(time, loc)_h$.

As before, the LocalCast service is parameterized by a radius, R. In order to implement the VMN Abstraction with an R-LocalCast communication service, we assume throughout this paper that the underlying model supports $2R$-LocalCast.

4 Solving Problems with Virtual Mobile Nodes

In this section we discuss some scenarios in which virtual nodes facilitate the design of algorithms, and discuss simple algorithms that address these situations. All the algorithms in this section depend only on the LocalCast service, although some may be simplified using a GeoCast service (as in [22, 5]), which delivers messages to a specific location (and any nodes that happen to be there).

Routing. We first consider the problem of routing messages. Neither the LocalCast service nor the GeoCast service delivers messages to specified nodes; both propagate messages to the mobile nodes in a well-defined region. Most algorithms that allow a client to explicitly specify a destination node (such as DSR [12, 13] and AODV [23]) are forced to either track the location of every mobile node, or flood the entire network with messages. Both approaches can be quite expensive, and optimizations are difficult.

The *PMN Routing Scheme* is based on the compulsory protocol of Chatzigiannakis [6]. In its simplest version, a single VMN travels through the network, collecting and delivering messages, as in Figure 2.a. (Pseudocode for such an algorithm is presented in Appendix A.) In order to send a message, a real node, i, examines its current location and calculates the current location of the virtual node that is carrying out the protocol. Node i then waits until the virtual node is nearby and sends it the message using the LocalCast service. The virtual node collects messages that it has received. Nodes that want to receive messages perform a similar protocol: if some real node, j, discovers that the VMN is nearby, it sends it a request for messages; if the VMN has any messages destined for j, it uses the LocalCast service to send them.

As presented, the algorithm has a low cost, where each message resides at only a single VMN; however a message might take a long time to be delivered. Using more VMNs may shorten the average message latency, while increasing the cost. For example, the VMNs might move as described in Figure 2.b. Whenever two VMNs pass each other, they send each other their stores of messages. In this way, all the messages spread to all the VMNs. A more space efficient algorithm might use the scheme developed in [6], where the VMNs form a snake, winding through the network in a pseudorandom path, thus regularly visiting every populated region of the network and delivering messages to the resident nodes.

Compared to typical routing schemes (such as DSR and AODV), these algorithms are easier to tune, in terms of space versus latency trade-offs: by increasing the number of VMNs, and thus the space, messages are more rapidly delivered. Unlike DSR and AODV, the cost (in terms of message latency and space usage) of these algorithms scales with the number of VMNs and the size of the region being covered, rather than the number of nodes.

Notice that the scheme described in Figure 2.b could also be used to implement a general GeoCast service using LocalCast. In this case, instead of each message eventually propagating to every VMN, a message only propagates to the VMN that travels along the horizontal coordinate specified in the message. Augmenting the system with vertical-traveling VMNs further increases the efficiency.

Since the motion of VMNs is known in advance, we can devise even more efficient algorithms for routing messages only among virtual nodes. The *VMN Routing Scheme* allows any VMN to send a message to any other VMN. (Pseudocode for such an algorithm is presented in Appendix A.) Consider sending a message, m, from the virtual node, S, to the virtual node, D. This task is performed by an additional *messenger* VMN, M. The node M ferries messages from source S to destination D (see Figure 2.c, where the solid lines represent the paths of the virtual nodes). When the node S is near M, it sends message m to M. When M is near D, it relays message m to D. (This is similar to a scheme presented in [4], that uses "buses" to carry data securely through a network.)

Although the paths intersect, we must ensure that the messenger actually meets nodes S and D. Therefore we choose the path of M such that the nodes meet periodically (for example, every time M intersect the path of S or D, it pauses until the expected node arrives).

The reliability of communication depends on the reliability of the messenger VMN. Moreover, since the paths of S, D, and M are predetermined, the message latency can be predicted.

Data Collection Algorithm. A common use of mobile *ad hoc* networks is to monitor environmental sensors. For example, one might wish to evaluate the average temperature, the average remaining battery charge, or the number of nodes in various regions of the network. (The latter application might be used to track animals or cars, or to determine the density of mobile nodes for other uses.)

Madden et al. [20] design a query language for sensor networks that specifies when, where, and how often data is sampled. They implement TinyDB, a sensor application that supports these queries. We propose a quite simple algorithm for addressing similar queries in mobile (rather than static) networks. Our algorithm also has the advantage of being easy to implement, with the potential to scale well. As in the case of routing, we direct the virtual nodes to systematically explore the region in question, collecting and aggregating data. This might use, for example, the pattern of motion described in Figure 2.b. A different topology might lead to different patterns of motion; for example, if the idea is to collect data about the goose population on the Charles River (a river separating Cambridge from Boston), the VMNs might be directed to all start at the west end of the river and sweep in a synchronized fashion to the east, ensuring that all the geese are discovered.

The primary difference from the routing algorithm is that the data may be aggregated, both regionally and temporally, as it is being collected. The client specifies in the query what sort of data the sensors should produce, and how the "network" should aggregate that data. The VMNs only return the necessary data to the clients. We present a simple version of this algorithm in Appendix C. (This version uses only a single VMN; as in the case of routing, multiple VMNs can improve the performance.)

Other Services. Many other basic services can be implemented using the VMN abstraction. For example, by extending the GeoQuorums algorithm [7], we can implement *Atomic Read/Write Memory*. The GeoQuorums algorithm depends on *focal points*, fixed, densely populated regions, to implement atomic objects that maintain replicas of the data. Instead, we suggest using VMNs to maintain the replicas. If certain areas of the network are known to be empty at certain times, then the replicas can be programmed to stay away from the dead spots. (For example, an office building may be empty at night; the replicas can be programmed to move in the evening to a nearby bar.) If areas of the network tend to issue many operations at certain times, the replicas can be programmed to move to these areas at these times, improving performance.

In the GeoQuorums algorithm, a GeoCast service is used to contact the replicas; in this case, the routing schemes discussed earlier in this section may be used. As in the GeoQuorums algorithm, the performance depends significantly on the distance of the replicas from the client and the efficiency of the GeoCast service that sends requests to the replicas. The VMN version should therefore be at least as efficient as the prior version; if the VMNs replicas are engineered to travel near the regions where most of the clients issue requests, it may be even more efficient.

Another example of a useful basic service is *Group Communication*. Dolev et al. [9] show how to implement group communication services (and resource allocation) using a single mobile agent that performs a random walk of a mobile *ad hoc* network. Unfortunately, this agent is a single point of failure. Using a reliable VMN to perform the data collection and propagation improves robustness. Also, by using a deterministic traversal, the VMN can visit the network more rapidly, thus improving performance.

5 Mobile Point Implementation of the VMN Abstraction

In this section, we present the *Mobile Point* algorithm, a robust implementation of the Virtual Mobile Node Abstraction. We first briefly consider a simple implementation that is not robust. We then discuss how the Mobile Point algorithm uses the LocalCast service to process messages in a consistent order, and finally present the Mobile Point Emulator, a robust implementation of the VMN abstraction. All the line numbers in this section refer to Figure 6.

Simple VMN Implementation. The *Agent Emulator* is a simple algorithm to implement fault-prone VMNs using *agents*. An agent is a dynamic process that jumps from one real node to another, moving in the direction specified by the VMN path. An agent "hitches a ride" with a host that is near to the specified location of the VMN. This strategy has been used in the past to implement various services, such as group communication (see [9]). It can be generalized to support arbitrary I/O automata. (We present code for the Agent Emulator in Appendix D.)

This simple algorithm meets one of the two goals of a VMN implementation: the movement of the virtual node is predictable. However, the host of the agent is a single point of failure, and therefore the VMN is not robust. For some applications, such as simple routing, this may be sufficient (as a higher level protocol can retry when messages are lost). For many applications, however, this lack of robustness is undesirable. We solve this problem through replication.

Mobile Points and Consistent Replication. We define a *mobile point* to be a circular region, of radius R, that moves along the same path as the virtual node: the center of the mobile point at time t is exactly the location of the virtual node at time t. At any given time, every node that resides within a mobile point replicates the virtual node.

Since the state of the virtual node is replicated at multiple nodes, the mobile point algorithm must maintain consistency among the replicas. We therefore use the LocalCast communication service and the synchronized clocks to implement a totally-ordered broadcast service within the region defined by a mobile point. We use a standard technique (derived from [16]) to ensure that each mobile node processes the messages in the same order: a timestamp is affixed to each message, defining a total order; before processing a message, the mobile node waits until at least time d has elapsed since the message was sent, ensuring that all earlier messages are received first. (See, for example, line 26.) The correctness of this scheme depends on reliable and timely message delivery. In Section 6, we discuss alternative algorithms for totally-ordered broadcast in which only liveness, and not correctness, depends on the underlying broadcast service.

The Mobile Point Emulator. We now present the Mobile Point Emulator algorithm. The algorithm is based on a replicated state machine technique (originally presented in [16]), augmented to support joins, leaves, and recovery. It uses the total ordering of messages to ensure that the replicas are updated consistently. The signature and state of the algorithm are given in Figures 3 and 4, and the algorithm itself is in Figure 6. All the code is presented using I/O Automata. See Lynch [19] for a full description of the IOA formalism.

Signature:

Input:
localcast-rcv$(m)_{h,i}$, $m \in M$
localcast-rcv$(\langle type, param, oid, \langle t, j\rangle\rangle)_{h,i}$
geo-update$(l, t)_{h,i}$, $l \in L$, $t \in R^{>0}$

Output:
localcast$(m)_{h,i}$, $m \in M$
localcast$(\langle type, param, oid, \langle t, j\rangle\rangle)_{h,i}$

Internal:
join$()_{h,i}$
leave$()_{h,i}$
reset$()_{h,i}$
simulate-action$(act)_{h,i}$, $act \in sig(\tau)$
simulate-geo-update$()_{h,i}$
process-join$()_{h,i}$
process-joinack$()_{h,i}$
process-reset$()_{h,i}$

Figure 3: MPE Signature for Node i and VMN h for I/O automaton $\tau = \langle sig, states, start, \delta\rangle$

Each real node executes one instance of the Mobile Point Emulator (MPE) for every VMN. The goal of the MPE is to replicate the state of the VMN at every node within the mobile point's region.

The $status$ of the Mobile Point Emulator can be one of four states: idle, indicating that the physical node is not within the area specified by the mobile point, joining or listening, indicating that the physical node is in the process of joining the mobile point, or active, indicating that the physical node is actively participating in the mobile point emulation.

When a node is active in the mobile point, it maintains a replicated copy of the state of the virtual node, val. This replicated state is modified only when the physical node receives a message indicating that a particular action should be performed. Since we ensure that all mobile nodes process the LocalCast messages in the same order, all nodes active in the emulation maintain consistent replicas. If the virtual node performs a localcast(m) action, then the real node itself sends m.

There are two scenarios that lead to the VMN taking a step of its computation, both of which begin with an active physical node sending a LocalCast message to the other replicas specifying the desired action. In the first scenario, an active node receives a message from a client to the VMN (lines 1–4). The physical node sends a LocalCast message to all the replicas, ensuring that all have received this message. Each replica, on receiving the rebroadcast (lines 20–35), updates the replicated state as if the VMN had performed a localcast-rcv action (lines 33–34).

In the second scenario, an active node may examine the current state of the VMN (i.e., val) and discover that some action is enabled, that is, ready to be performed (lines 12–18). The replica then sends a broadcast to all the other replicas instructing them to take a step of the computation. When an active node receives

State:
$status \in$ {idle,joining,listening,active}, initially active if node is in $mp\text{-}location$ and idle otherwise
$val \in states(\tau)$, holds current state of the simulated I/O automaton, initially start(τ)
msg-queue, a queue of messages to be sent by the communication service, initially $\emptyset$
anwered-join-reqs, set of ids of join requests that have already been answered, initially $\emptyset$
join-id $\in T$, unique id for current join request, initially $\langle 0, i_0\rangle$
pending-actions, set of messages waiting to be processed, initially $\emptyset$
completed-actions, queue of actions that have been simulated, initially $\emptyset$
mp-location $\in L$, continuous, location defining the center of the mobile point under consideration
clock $\in R^{\geq 0}$, the current time, initially 0, continuously updated by the Geosensor
location $\in L$, the current physical location, continuously updated by the Geosensor

Figure 4: MPE State for Node i and VMN h for I/O automaton $\tau = \langle sig, states, start, \delta\rangle$

such an instruction, it performs the step of the computation, if the action is still valid (lines 20–35).

Joining a Mobile Point. Whenever a physical node is within the perimeter defined by the mobile point, it initiates a join protocol (lines 65–72); whenever a physical node is outside of a mobile point, it executes the leave protocol, which reinitializes its states and sets its status to idle (lines 100–112). The maximum speed of the mobile point is determined by the latency of the join protocol and the speed of the real physical nodes: the mobile point must move slowly enough so that new nodes can enter and join the mobile point before the old nodes leave.

Assume that node i is attempting to join. The join protocol begins when i broadcasts a join-req to all the replicas requesting a copy of the current state. When node i receives its own join request, it enters the listening state (lines 81–82). This indicates that it can begin to monitor the messages in the system. In particular, it saves any LocalCast messages that temporally follow its join request in $pending\text{-}actions$.

When some other active node, j, receives the join request, it sends a join acknowledgment, join-ack. This acknowledgment includes a copy of its replica of the virtual node, val_j. When i receives the acknowledgment, it makes a copy of the replicated state, and begins to process its $pending\text{-}actions$ (lines 87–98). Notice that by the time i receives the copy of the replicated state, the copy may be out of date. Fortunately, the messages stored in $pending\text{-}actions$ are sufficient to bring val_i up to date.

Recovery. A mobile point may fail when it reaches a depopulated area of the network. As soon as all the nodes leave the mobile point, it loses its state. The Mobile Point Emulator contains a recovery mechanism that allows the virtual node to be restarted in this case. When a real node enters the mobile point and cannot communicate with any other active nodes, it can broadcast a reset message (lines 43–47); every node receiving a recover message reinitializes its state (lines 49–63). If multiple nodes try to initiate a recovery at the same time, the last node to send the recover messages begins the recovery.

Correctness. We claim that the Mobile Point Emulator correctly implements the Virtual Mobile Node Abstraction, in that any service built on the VMN abstraction runs correctly on the Mobile Point Emulator:

Theorem 5.1 *Let A be a system consisting of client automata and a virtual node, where all* localcast *and* localcast-rcv *actions are hidden. Let S be a system consisting of client automata composed with the Mobile Point Emulator, which implements the virtual node. Again, hide the* localcast *and* localcast-rcv *actions in S. Then $traces(S) \subseteq traces(A)$.*

VMN Liveness. The Mobile Point algorithm ensures that the VMN fails only when it enters a depopulated region of the network. Therefore, while we make no assumptions about the motion of the real mobile nodes, we do assume that certain (time varying) regions are usually "populated" by real nodes. That is, we assume that at all times, at least one node resides in the region defined by a ball of radius R around the location of the virtual node: there is always at least one real node in the mobile point. Further, we assume that real nodes entering and leaving the mobile point overlap in time for "sufficiently long", i.e., enough time for the departing real nodes to transmit the state to the entering nodes.

If the mobile point becomes depopulated at any point, the virtual node fails. If it becomes repopulated at a later point, however, it can be restored, beginning again in its initial state.

We claim that, in the real world, this density assumption is reasonable. There often exist regions that are almost always populated — such as highways, office buildings, and shopping malls. Furthermore, these areas tend to have predictable patterns of density. For example, a mobile point that spends its days exploring the highways may move to a more populated area at night.

6 Discussion and Concluding Remarks

The VMN framework, and the algorithms presented in this paper, introduce new horizons for further research. One significant path of investigation is to devise further applications for the new robust primitive presented in this paper: the algorithms given in Section 4 only begin to examine the possible uses of this abstraction. And it will be quite interesting to experiment with a real implementation of VMNs, to determine if the real utility of VMNs outweighs the overhead of implementing them. We focus in this section on a second area of ongoing investigation: alternate implementations of virtual mobile nodes. We consider four improvements: (i) more dynamic VMNs, (ii) self-stabilizing VMN implementations, (iii) a VMN implementation for a more asynchronous environment, and (iv) combining the Mobile Point and Agent algorithms to implement an even more efficient VMN.

Dynamic Virtual Mobile Nodes. In this paper we have assumed that the path of a VMN is fixed in advance, and the set of VMNs is fixed in advance. In many cases, this is not only sufficient, but in fact advantageous, as the location of a VMN can be known *a priori*. However, for some applications it would be useful if the path of the VMN could be determined on-the-fly. For example, one can imagine using a VMN to follow a user, either performing a service for that user, or tracking the location of the user. In addition, the dynamic path could actively avoid unpopulated areas of the network, thus improving robustness. Similarly, it may be useful to generate virtual nodes dynamically. Consider, for example, the security example where a new VMN is generated to track every intruder that enters a certain area.

Self-Stabilizing Mobile Points. Long-term robustness of the VMN abstraction can be improved if the virtual nodes were self-stabilizing. We believe that with a few modifications, the Mobile Point algorithm can be made self-stabilizing.

Broadcast in the Partially Synchronous Model. The correctness of the Mobile Point Emulator depends on the reliable and timely delivery of broadcast messages. There are a number of algorithms, however, for performing totally-ordered broadcast in partially synchronous environments (for example, [17]). In these algorithms, correctness does not depend on any timing assumptions; instead, the eventual delivery of messages depends on eventually timely broadcast. We believe it possible to implement a VMN using such an algorithm (in particular, a variant of an algorithm presented in [1]). Alternatively, there may be other practical ways to implement a sufficient broadcast service. (One other option is discussed in [7], for example.)

Mobile Points and Agents. Finally, we observe that it may improve performance to execute the Mobile Point Emulator on virtual nodes implemented by unreliable agents, rather than on real nodes directly. The agents are quite simple and efficient, but unreliable; the mobile point emulator is reliable, but not as efficient. By combining the two algorithms, we can improve the efficiency without sacrificing the reliability.

Conclusions In this paper, we have presented a new technique for implementing algorithms in mobile *ad hoc* networks. In general, it is quite difficult to devise algorithms for such a chaotic, unpredictable environment. In the VMN Abstraction, however, there exist reliable (virtual) nodes that move in a predictable manner. Algorithms running on virtual nodes can worry less about fault-tolerance, and more about solving the task at hand. And algorithms running on virtual nodes can be assured that the virtual node will not be confined to undesirable parts of the network. Using only local communication, a VMN is able to participate in global network affairs. We have presented the Mobile Point Emulator, a new algorithm that takes advantage of location information and dense areas of the network to implement reliable virtual nodes. We believe that this abstraction, and algorithms in the paradigm of the Mobile Point Emulator, will significantly simplify the development of algorithms for mobile networks.

```
Input localcast-rcv(m)_{h,i}
Effect:
    if (m ∉ pending-actions ∪ completed-actions) then
        Enqueue(msg-queue, ⟨simulate, ⟨rcv, m⟩, ⊥⟩)

Input localcast-rcv(⟨type, param, oid, ⟨t, j⟩⟩)_{h,i}
Effect:
    if (⟨type, param, oid, ⟨t, j⟩⟩ ∉ pending-actions) ∧
        (⟨type, param, oid, ⟨t, j⟩⟩ ∉ completed-actions)) then
        pending-actions ← pending-actions ∪ {⟨type, param, oid, ⟨t, j⟩⟩}

Internal init-action(act)_{h,i}
Precondition:
    status = active
    δ(act) ≠ ⊥
Effect:
    oid ← ⟨time, i⟩
    Enqueue(msg-queue, ⟨simulate, act, oid⟩)

Internal simulate-action(act)_{h,i}
Precondition:
    status ≠ listening
    ⟨simulate, act, oid, ⟨t, j⟩⟩ ∈ pending-actions
    ∀⟨∗, ∗, ∗, ⟨t′, j′⟩⟩ ∈ pending-actions:
        (t < t′) ∨ ((t = t′) ∧ (j < j′))
    t+d ≥ time
Effect:
    if (status = active) then
        if (δ(val, act) ≠ ⊥)then
            val ← δ(val, act)
        completed-actions ← completed-actions ∪
                                {⟨simulate, act, oid, ⟨t, j⟩⟩}
        if (act = ⟨send, m⟩) then
            Enqueue(msg-queue, m)
    pending-actions ← pending-actions \ {⟨simulate, act, oid, ⟨t, j⟩⟩}

Internal simulate-geo-update()_{h,i}
Precondition:
    None.
Effect:
    val ← δ(val, ⟨geo-update, clock, mp-location⟩)

Internal reset()_{h,i}
Precondition:
    status = listening
Effect:
    Enqueue(msg-queue, ⟨reset, ⊥, ⊥⟩)

Internal simulate-reset()_{h,i}
Precondition:
    ⟨reset, ∗, ∗, ⟨t, j⟩⟩ ∈ pending-actions
    ∀⟨∗, ∗, ∗, ⟨t′, j′⟩⟩ ∈ pending-actions:
        (t < t′) ∨ ((t = t′) ∧ (j < j′))
    t+d ≥ time
Effect:
    status ← active
    val ← start(τ)
    join-id ← ⟨0, i_0⟩
    answered-join-reqs ← ∅
    pending-actions ← ∅
    completed-actions ← ∅
    lbcast-queue ← ∅
    geocast-queue ← ∅

Internal join()_{h,i}
Precondition:
    | location -mp-location| ≤ R
    status = idle
Effect:
    join-id ← ⟨clock, i⟩
    status ← joining
    Enqueue(msg-queue, ⟨join-req, ⊥, join-id⟩)

Internal process-join()_{h,i}
Precondition:
    ⟨join-req, ∗, jid, ⟨t, j⟩⟩ ∈ pending-actions
    ∀⟨∗, ∗, ∗, ⟨t′, j′⟩⟩ ∈ pending-actions:
        (t < t′) ∨ ((t = t′) ∧ (j < j′))
    t+d ≥ time
Effect:
    if ((status = joining) ∧ (jid = join-id)) then
        status ← listening
    if ((status = active) ∧ (jid ∉ answered-join-reqs)) then
        Enqueue(msg-queue, ⟨join-ack, jid, val⟩)
    pending-actions ← pending-actions \ {⟨join-req, ⊥, jid, ⟨t, j⟩⟩}

Internal process-joinack()_{h,i}
Precondition:
    ⟨join-req, v, jid, ⟨t, j⟩⟩ ∈ pending-actions
    ∀⟨join-req, ∗, ∗, ⟨t′, j′⟩⟩ ∈ pending-actions:
        (t < t′) ∨ ((t = t′) ∧ (j < j′))
    t+d ≥ time
Effect:
    answered-join-reqs ← answered-join-reqs ∪ {jid}
    if ((status = listening) ∧ (jid = join-id)) then
        status ← active
        val ← v
    pending-actions ← pending-actions \ {⟨join-act, v, jid, ⟨t, j⟩⟩}

Internal leave()_{h,i}
Precondition:
    | location -mp-location| > R
    status ≠ idle
Effect:
    status ← idle
    join-id ← ⟨0, i_0⟩
    val ← start(τ)
    answered-join-reqs ← ∅
    pending-actions ← ∅
    completed-actions ← ∅
    lbcast-queue ← ∅
    geocast-queue ← ∅

Output localcast(m)_{h,i}
Precondition:
    Peek(msg-queue) = m
Effect:
    Dequeue(msg-queue)

Output localcast(⟨type, param, oid, ⟨t, i⟩⟩)_{h,i}
Precondition:
    Peek(msg-queue) = ⟨type, param, oid⟩
    t = time
Effect:
    Dequeue(msg-queue)

Input geo-update(l, t)_{h,i}
Effect:
    location ← l
    clock ← t
```

Figure 6: Mobile Point that implements VMN h executing I/O automaton $\tau = \langle sig, states, start, \delta \rangle$

References

[1] BAR-JOSEPH, Z., KEIDAR, I., AND LYNCH, N. A. Early-delivery dynamic atomic broadcast. In *Proceedings of the 16th International Symposium on DIStributed Computing* (2002), pp. 1–16.

[2] BEAL, J. Persistent nodes for reliable memory in geographically local networks. Tech. Rep. AIM-2003-11, MIT, 2003.

[3] BEAL, J. A robust amorphous hierarchy from persistent nodes. In *Proceedings of the International Conference on Communication Systems and Networks* (2003).

[4] BEIMEL, A., AND DOLEV, S. Buses for anonymous message delivery. *Journal of Cryptology 16*, 1 (2003), 25–29.

[5] CAMP, T., AND LIU, Y. An adaptive mesh-based protocol for geocast routing. *Journal of Parallel and Distributed Computing: Special Issue on Mobile Ad-hoc Networking and Computing* (2002), 196–213.

[6] CHATZIGIANNAKIS, I., NIKOLETSEAS, S., AND SPIRAKIS, P. An efficient communication strategy for ad-hoc mobile networks. In *Proc. 15th International Symposium on Distributed Computing* (2001), pp. 320 – 322.

[7] DOLEV, S., GILBERT, S., LYNCH, N. A., SHVARTSMAN, A. A., AND WELCH, J. Geoquorums: Implementing atomic memory in mobile ad hoc networks. In *Proceeding of the 17th International Conference on Distributed Computing* (October 2003).

[8] DOLEV, S., GILBERT, S., LYNCH, N. A., SHVARTSMAN, A. A., AND WELCH, J. L. Geoquorums: Implementing atomic memory in ad hoc networks. Tech. Rep. LCS-TR-900, MIT, 2003.

[9] DOLEV, S., SCHILLER, E., AND WELCH, J. Random walk for self-stabilizing group communication in ad-hoc networks. In *Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems* (2002), pp. 70–79.

[10] *Communications of the ACM, Special section on Group Communication Systems* (1996), vol. 39(4).

[11] HATZIS, K. P., PENTARIS, G. P., SPIRAKIS, P. G., TAMPAKAS, V. T., AND TAN, R. B. Fundamental control algorithms in mobile networks. In *Proc. of the 1th ACM symposium on Parallel Algorithms and Architectures archive* (Saint Malo, France, 1999), pp. 251 – 260.

[12] JOHNSON, D. B., AND MALTZ, D. A. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, T. Imielinski and H. Korth, Eds. Kluwer Academic Publishers, 1996, ch. 5, pp. 153–181.

[13] JOHNSON, D. B., MALTZ, D. A., AND BROCH, J. DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks. *Ad Hoc Networking* (2001), 139–172.

[14] KEIDAR, I. A highly available paradigm for consistent object replication. Master's thesis, Hebrew University, Jerusalem, 1994. URL: http://www.cs.huji.ac.il/simtransis/publications.html.

[15] KEIDAR, I., AND DOLEV, D. Efficient message ordering in dynamic networks. In *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing* (1996), ACM Press, pp. 68–76.

[16] LAMPORT, L. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM 21*, 7 (july 1978), 558–565.

[17] LAMPORT, L. The part-time parliament. *ACM Transactions on Computer Systems 16*, 2 (1998), 133–169.

[18] LI, Q., AND RUS, D. Sending messages to mobile users in disconnected ad-hoc wireless networks. In *Proceedings of the 6th Annual ACM/IEEE International Conference on Mobile Computing* (2000).

[19] LYNCH, N. A. *Distributed Algorithms*. Morgan Kaufman, 1996.

[20] MADDEN, S., FRANKLIN, M. J., HELLERSTEIN, J. M., AND HONG, W. The design of an acquisitional query processor for sensor networks. In *SIGMOD Conference* (2003), pp. 491 – 502.

[21] NATH, B., AND NICULESCU, D. Routing on a curve. *ACM SIGCOMM Computer Communication Review 33*, 1 (January 2003), 150 – 160.

[22] NAVAS, J. C., AND IMIELINSKI, T. Geocast – geographic addressing and routing. In *ACM/IEEE Intl. Conference on Mobile Computing and Networking* (1997), pp. 66–76.

[23] PERKINS, C., AND ROYER, E. M. Ad-hoc on-demand distance vector (AODV) routing. In *Proceedings of IEEE WMCSA* (1999), pp. 90–100.

[24] PRIYANTHA, N. B., CHAKRABORTY, A., AND BALAKRISHNAN, H. The cricket location-support system. In *Proc. of the 6th ACM MOBICOM* (August 2000), pp. 32–43.

A Physical Mobile Node Routing Scheme

State:
id, constant, physical mobile node identifier
time, continuously changing, the current time
location, continuously changing, the current location of VMN i
messenger-location, continuously changing, the current location of the messenger VMN
local-messages, a set of messages to be forwarded, initially $\emptyset$.
delivered, a set of messages to deliver, initially $\emptyset$

Input send$(m)_{i,j}$
Effect:
$local\text{-}messages \leftarrow local\text{-}messages \cup \langle i,j,m \rangle$

Output rcv$(m)_{j,i}$
Precondition:
$m \in delivered$
Effect:
$delivered \leftarrow delivered \setminus \{m\}$

Output localcast(message, $m)_i$
Precondition:
$m \in local\text{-}messages$
$|\ messenger\text{-}location - location|\ \leq R$
Effect:
$local\text{-}messages \leftarrow local\text{-}messages \setminus \{m\}$

Output localcast(nbr, i, t, $\ell)_i$
Precondition:
$|\ messenger\text{-}location - location|\ \leq R$
$t = time$
$\ell = location$
Effect:
None.

Input localcast-rcv(message, $\langle src, dest, m \rangle)_{j,i}$
Effect:
if $(dest = i)$ **then**
$delivered \leftarrow delivered \cup \{m\}$

IOA code for running on PMN i that wants to send and receive messages. The input send and the output rcv are the external interface to the routing service. The PMN notifies the messenger VMN of itself with a nbr message, and sends it messages to deliver.

State:
id, constant, physical mobile node identifier
timeout, constant, frequency with which broadcasts occur
time, the current real time, as of the last Geosensor update
location, the current location, as of the last Geosensor update
nbr-loc, set of neighbors, tuples of the form $\langle id, time, location \rangle$, initially $\emptyset$
local-messages, a set of messages to be forwarded, initially $\emptyset$.
delivered, a set of messages to deliver, initially $\emptyset$

Input localcast-rcv(message, $m)_{j,i}$
Effect:
$local\text{-}messages \leftarrow local\text{-}messages \cup m$

Input localcast-rcv(nbr, id, $time$, $location)_{j,i}$
Effect:
$nbr\text{-}loc \leftarrow nbr\text{-}loc \cup \langle id, time, location \rangle$

Output localcast(message, $\langle src, dest, m \rangle)_{i,j}$
Precondition:
$\langle j, t, \ell \rangle \in nbr\text{-}loc$
$(R - |\ location - \ell|\ ) \geq (v_{max} \cdot |\ time - t|\ )$
$j = dest$
$\langle src, dest, m \rangle \in local\text{-}messages$
Effect:
$local\text{-}messages \leftarrow local\text{-}messages \setminus \langle src, dest, m \rangle$

IOA code for running on messenger VMN i. The VMN listens for nearby nodes (i.e., nbr messages), and delivers messages.

B Virtual Mobile Node Routing Scheme

State:
id, constant, physical mobile node identifier
location, continuously changing, the current location of VMN i
vmn-location(ℓ), continuously changing, the current location of VMN ℓ
local-messages, a set of messages to be forwarded, initially $\emptyset$.
delivered, a set of messages to deliver, initially $\emptyset$

Input send$(m)_{i,j}$
Effect:
$local\text{-}messages \leftarrow local\text{-}messages \cup \langle i,j,m \rangle$

Output rcv$(m)_{j,i}$
Precondition:
$m \in delivered$
Effect:
$delivered \leftarrow delivered \setminus \{m\}$

Output localcast(message, $m)_i$
Precondition:
$m \in local\text{-}messages$
$\langle$, **dest,** $\rangle = m$
$\ell = next\text{-}hop(i, dest)$
$|\ vmn\text{-}location(\ell) - location\} \leq R$
Effect:
$local\text{-}messages \leftarrow local\text{-}messages \setminus \{m\}$

Input localcast-rcv(message, $\langle src, dest, m \rangle)_{j,i}$
Effect:
if $(dest = i)$ **then**
$delivered \leftarrow delivered \cup \{m\}$
else
$local\text{-}messages \leftarrow local\text{-}messages \cup \{\langle src, dest, m \rangle\}$

IOA code for running on VMN that wants to send and deliver messages. The *next-hop* function determines the next hop in the routing path, determined by running a time-dependent shortest-path algorithm on the motion paths of the VMNs. Each VMN calculates the next hop, and waits until it is near the desired VMN. At this point, it passes the message to that virtual node.

C Simple Data Collection Scheme

State:
id, constant, physical mobile node identifier
result-set, current set of results, initially $\emptyset$
data-set, current raw data set, initially $\emptyset$
outgoing, set of outgoing messages, initially $\emptyset$
q, a query structure, with the following subfields:
area, the geographic area in which the query should be evaluated
freq, the frequency with which the data should be sampled
duration, the duration with which the data should be sampled
eval(), an evaluation procedure for the query
comb(...), procedure that combines two data sets

Input localcast-rcv(query, *new-query*)$_{j,i}$
Effect:
$\textit{new-query} \leftarrow q$

Input localcast-rcv(data, *new-data*)$_{j,i}$
Effect:
If data, location, and time meet the requirements of the query, then:
$\textit{data-set} \leftarrow q.\textit{comb}(\textit{data-set}, \textit{new-data})$

Input localcast-rcv(request-data)$_{j,i}$
Effect:
$\textit{outgoing} \leftarrow \textit{outgoing} \cup \{\langle \text{output-data}, \textit{data-set}\rangle\}$

Output localcast(query, q)$_{i,j}$
Precondition:
None.
Effect:
None.

Output localcast(m)$_{j,i}$
Precondition:
$m \in \textit{outgoing}$
Effect:
$\textit{outgoing} \leftarrow \textit{outgoing} \setminus \{m\}$

Internal aggregate()$_i$
Precondition:
None.
Effect:
$\textit{result-set} \leftarrow q.\textit{eval}(\textit{data-set})$

IOA code for running on VMNs collecting and processing sensor data. The VMN spreads the query to all the PMNs, and collects and aggregates data from the PMNs. When it receives a request for data, it sends it out.

State:
id, constant, physical mobile node identifier
sensor-data, continously updated sensor data
VMN-locations(ℓ), continuously updated, the locatino of VMN ℓ
data-set, current accumulated data, initially $\emptyset$
outgoing, a set of outgoing localcast messages, initially $\emptyset$
virtual-nodes, set of virtual nodes collecting data, initially $\emptyset$
q, a query structure, with subfields as above

Input initiate-query(*new-query*)
Effect:
$\textit{outgoing} \leftarrow \textit{outgoing} \cup \{\langle \text{query}, \textit{new-query}\rangle\}$

Input localcast-rcv(query, *new-query*)$_{j,i}$
Effect:
$q \leftarrow \textit{new-query}$
$\textit{virtual-nodes} \leftarrow \textit{virtual-nodes} \cup \{j\}$

Input localcast-rcv(output-data, *new-data-set*)
Effect:
Send data to requester

Output localcast(data, *data-set*)$_{i,j}$
Precondition:
$j \in \textit{virtual-nodes}$
$|\ \textit{VMN-locations}(j) - \textit{location}|\ \leq R$
Effect:
None.

Output localcast(request-data)
Precondition:
Node i wants data
Effect:
None.

Output localcast(m)$_{j,i}$
Precondition:
$m \in \textit{outgoing}$
$|\ \textit{VMN-locations}(j) - \textit{location}|\ \leq R$
Effect:
$\textit{outgoing} \leftarrow \textit{outgoing} \setminus \{m\}$

Internal store-data()$_i$
Precondition:
$location$ is consistent with $q.area$
$time$ is consistent with $q.freq$ and $q.duration$
Effect:
$\textit{data-set} \leftarrow q.\textit{comb}(\textit{data-set}, \textit{sensor-data})$

IOA code for running on sensor PMNs. When the sensor PMN wants to initiate a new query, it sends the query to the nearest VMN. The VMN then propagates the query to all the sensors. Each sensor then stores and aggregates the data, and sends it on to a passing VMN.

D Agent Emulator Implementation

Signature:

Input:
$\mathsf{rcv}(m)_{h,i}$, $m \in M$
$\mathsf{rcv}(\langle\mathsf{neighbor\text{-}update}, \mathit{nbr\text{-}loc}, \mathit{src}\rangle)_i$, $\mathit{nbr\text{-}loc} \in \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0}$
$\mathsf{rcv}(\langle\mathsf{agent}, v, \mathit{dest}\rangle)_i$, $v \in \mathit{states}(\tau)$, $\mathit{dest} \in I$
$\mathsf{geo\text{-}update}(l, t)_{obj,i}$, $l \in L$, $t \in R^{>0}$

Output:
$\mathsf{send}(m)_{h,i}$, $m \in M$
$\mathsf{send}(\langle\mathsf{neighbor\text{-}update}, \mathit{nbr\text{-}loc}\rangle)_i$, $\mathit{nbr\text{-}loc} \in \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0}$
$\mathsf{send}(\langle\mathsf{agent}, v, \mathit{dest}\rangle)_i$, $v \in \mathit{states}(\tau)$, $\mathit{dest} \in I$

Internal:
$\mathsf{simulate\text{-}action}()_{h,i}$
$\mathsf{simulate\text{-}geo\text{-}update}()_{h,i}$

State:

agent-location $\in L$, continuous, the location of the VMN at the current time
clock $\in R^{\geq 0}$, the current time, initially 0, continuously updated by the geosensor
location $\in L$, the current physical location, continuously updated by the geosensor
msg-queue, a queue of messages to be sent by the communication service, initially $\emptyset$
nbr-set, location information on neighbors, initially $\emptyset$
status $\in \{\mathsf{active}, \mathsf{idle}\}$, indicates whether the host is emulating the VMN
val $\in$ *states*(τ), holds current state of the simulated I/O automaton, initially start(τ)

Transitions:

Input $\mathsf{rcv}(m)_{h,i}$
Effect:
if $(\mathit{status} = \mathsf{active})$
$\mathit{val} \leftarrow \delta(\mathit{val}, \langle\mathsf{rcv}, m\rangle)$

Input $\mathsf{rcv}(\langle\mathsf{neighbor\text{-}update}, \mathit{nbr\text{-}loc}, \mathit{src}\rangle)_i$
Effect:
$\mathit{nbr\text{-}set} \leftarrow \mathit{nbr\text{-}set} \cup \{\ \langle \mathit{src}, \mathit{nbr\text{-}loc}, \mathit{clock}\rangle\ \}$

Input $\mathsf{rcv}(\langle\mathsf{agent}, v, \mathit{dest}\rangle)_i$
Effect:
if $(\mathit{dest} = i)$ **then**
$\mathit{val} \leftarrow v$

Input $\mathsf{geo\text{-}update}(l, t)_{h,i}$
Effect:
$\mathit{location} \leftarrow l$
$\mathit{clock} \leftarrow t$

Internal $\mathsf{simulate\text{-}action}()_{h,i}$
Precondition:
$\mathit{status} = \mathsf{active}$
$\delta(\mathit{val}, \mathit{act}) \neq \perp$
$\mathit{act} \neq \langle\mathsf{send}, m\rangle$
Effect:
$\mathit{val} \leftarrow \delta(\mathit{val}, \mathit{act})$

Output $\mathsf{send}(m)_{h,i}$
Precondition:
$\mathit{status} = \mathsf{active}$
$\delta(\mathit{val}, \langle\mathsf{send}, m\rangle) \neq \perp$
Effect:
$\mathit{val} \leftarrow \delta(\mathit{val}, \langle\mathsf{send}, m\rangle)$

Output $\mathsf{send}(\langle\mathsf{neighbor\text{-}update}, \mathit{nbr\text{-}loc}, \mathit{src}\rangle)_i$
Precondition:
$\mathit{nbr\text{-}loc} = \mathit{location}$
$\mathit{src} = i$
Effect:
None.

Output $\mathsf{send}(\langle\mathsf{agent}, v, \mathit{dest}\rangle)_i$
Precondition:
$\mathit{status} = \mathsf{active}$
$\langle \mathit{dest}, \ell, t\rangle \in \mathit{nbr\text{-}set}$
$R - |location - \ell| \geq v_{max} \cdot |clock - t|$
$R - |agent\text{-}location - \ell| \geq V \cdot |clock - t|$
Effect:
$\mathit{status} \leftarrow \mathsf{idle}$

Internal $\mathsf{simulate\text{-}geo\text{-}update}()_{h,i}$
Precondition:
None.
Effect:
$\mathit{val} \leftarrow \delta(\mathit{val}, \langle \mathit{geo\text{-}update}, \mathit{clock}, \mathit{agent\text{-}location}\rangle)$

Agent Emulator Implementation for Node i and VMN h for I/O automaton $\tau = \langle sig, states, start, \delta\rangle$