

# An Entity Maintenance and Connection Service for Sensor Networks

Brian Blum, Prashant Nagaraddi, Anthony Wood, Tarek Abdelzaher, Sang Son, Jack Stankovic  
*Department of Computer Science*  
*University of Virginia, Charlottesville, VA 22904*

## Abstract

*In this paper, we present a middleware architecture for coordination services in sensor networks that facilitates interaction between groups of sensors which monitor different environmental events. It sits on top of the native routing infrastructure and exports the abstraction of mobile communication endpoints maintained at the locations of such events. A single logical destination is created and maintained for every environmental event of interest. Such destinations are uniquely labeled and can be used for communication by application-level algorithms for coordination and sensory data management between the different event locales. For example, they may facilitate coordination, in a distributed intrusion scenario, among nodes in the vicinity of the intruders.*

*We evaluate our middleware architecture using GloMoSim, a wireless network simulator. Our results illustrate the success of our architecture in maintaining event-related communication endpoints. We provide an analysis of how architectural and network dependent parameters affect our performance. Additionally we provide a proof of concept implementation on a real sensor network testbed (Berkeley's MICA Motes).*

## 1 Introduction

The impending proliferation of sensor networks calls for new services and middleware for distributed deeply embedded computing. We consider ad hoc networks formed by dropping wireless computationally-equipped sensors (e.g., from an airplane) onto a dangerous or inaccessible infrastructureless environment such as a disaster area or a hostile territory behind enemy lines. The lack of an infrastructure implies that no workstations or other centralized computing equipment are present for management or information processing purposes. Such sensor networks will therefore have to host their own distributed embedded computation. They will execute their mission autonomously while interacting with spatially distributed external events in the physical environment. In this paper, an environmental event refers to an

ongoing activity, such as the motion or presence of a vehicle, that persists in the physical world for some continuous interval of time.

A new distributed computing paradigm is needed to support the writing and execution of distribution applications for such networks. Due to the tight coupling between computation in the sensor network and events in the environment, one key requirement of such a paradigm is to support applications that coordinate teams of sensor and actuator nodes in the vicinity of different external events of interest. For example, one may want to exchange data among all nodes in the vicinity of hostile targets in the sensor field to determine a plan of attack.

The aforementioned coordination problem offers an interesting research challenge to the communication subsystem pertaining to mobility. In PDAs and Wireless LANs, supporting mobility typically refers to maintaining connectivity between individual devices despite their changing spatial relationship to one another. In contrast, what moves in our scenario are the external environmental events. Sensor network nodes themselves remain relatively motionless. In this paper, we present middleware that allows programmers to think of mobile external events in terms of abstract persistent entities that logically form in the network as a response to appropriate sensor readings. The middleware forms and maintains a unique entity around each event. These entities are addressable and act as communication destinations (end-points). Note that we define an *event* as something in the environment that causes a sensor to report a certain reading (e.g., fire, moving vehicle, etc.) and an *entity* as the abstract addressable equivalent of this event as referenced by the programmer.

The communication problem is therefore to maintain the abstraction of transport-layer connections between different entities, when each entity is composed of a changing set of sensor nodes at the location of a mobile external event. Such mapping is made complicated by several factors. One is the need for seamless end-point migration across nodes as the event moves. Another is that sensor nodes that become aware of an external event should be able to decide whether it is the same event previously seen by other sen-

sors or a new event. Otherwise, an incorrect event list will be collectively maintained or an incorrect mapping will result between events and communication endpoints. This paper reports a communication architecture which resolves these challenges.

Our middleware hides the details of sensor group formation around environmental events, end-to-end connection establishment between different entities, and entity maintenance to ensure that a single abstract entity is created and maintained for every event of interest in the environment. This architecture's ability to ensure a one to one relationship between abstract entities and environmental events simplifies communication, facilitates coordination, and reduces programming complexity by providing communication with persistent entities instead of dynamically changing sets of individual nodes.

Our techniques are geared for the case where events are reasonably sparse (i.e., the tracked environmental targets are generally not in close proximity). Disambiguating nearby targets is an inherently difficult problem that is not addressed in this paper. The reader may want to think of our architecture as imposing a resolution constraint. Targets that are closer than the available resolution cannot be individually distinguished. As we shall see later, the resolution is of the order of the communication radius of the sensor nodes. In the physical sensor node prototype available to the authors this radius can be adjusted from several inches to hundreds of feet.

The remainder of this paper is organized as follows. We review related work in Section 2. An overview of entity establishment and maintenance as well as details of the underlying protocols follows in Section 3. Detailed simulation results are discussed in Section 4. A description of an implemented proof-of-concept prototype is given in Section 5. Finally, we explore future work and conclude in Section 6.

## 2 Related Work

Sensor networks [11] have recently emerged as a promising platform for a myriad of distributed embedded applications in defense [42] and scientific exploration [21]. A typical sensor network is highly distributed and composed of thousands to hundreds of thousands of individual nodes. Communication protocols are therefore a very important research topic in sensor networks.

Most prior work on communication in sensor networks has focused on the lower layers in the protocol stack. For example, [41, 45] propose MAC layer protocols designed for sensor networks. At the network layer, protocols such as DSDV [27], DSR [14], AODV [29] and TORA [25] have gained popularity as routing solutions for ad hoc wireless networks. These protocols are designed for networks with

identifier-based node addressing. Recent sensor network research suggests alternative addressing schemes that do not rely on having destinations with specific identities. Instead, it has been proposed that routing in sensor networks be attribute-based where the destination is reached by its attributes such as location or sensor measurements. For example, LAR [17] and DREAM [3] propose location-aware routing protocols, where the destination is implicitly defined by its physical location. Directed diffusion [13] and the intentional naming system [1] provide routing and addressing based on data interests. A related effort is attribute-based naming [32], proposed for an Internet environment, which allows queries to be routed depending on the requested content rather than on the identity of the target machine. Our work falls in the general category of attribute-based communication. We provide an infrastructure where communication end-points are placed at the locations of specific events in the environment. Unlike prior work on attribute-based addressing, we focus on protocol dynamics that arise due to the motion of such events in the external world. We aim to maintain the persistence and uniqueness of these communication end-points as the event moves and discuss factors that affect the maximum trackable speed of these events. Also, unlike routing-layer approaches, our architecture sits in the transport layer on top of geographic forwarding.

Mobile end-points have been addressed in traditional and ad hoc computer networks. For example, Energy-aware routing protocols were proposed such as Span [6] and GAF [43] for communication between mobile nodes. In Mobile IP [26], mobile hosts are free to migrate between LAN's while they remain connected by a home agent residing at their home address. Another mechanism for maintaining mobile connections uses DNS to provide the indirection necessary to support mobility [34]. We address mobility in a different sense in our protocol. Whereas the aforementioned protocols assume moving nodes and provide for communication between these moving nodes, we assume static nodes but moving events (and thus entities) in our system and provide a migratory end-point infrastructure for communication between these moving entities. A recent protocol, called TTDD [44], addresses communication between moving sources and sinks in a sensor network. Our work differs from this in the fact that we provide for the creation and maintenance of abstract entities to facilitate communication between moving events in the network. Also, our communication end-points are bi-directional as opposed to being statically designated as sources or sinks.

Several algorithms exist that provide clustering and various granular levels of group formation on both the network and application layers. The  $(\alpha, t)$  framework [22, 37], and Landmark Hierarchy [39] organize nodes into hierarchical groups as a solution to routing. LEACH [10], ASCENT

[5], SPAN [6], and GAF [43] form groups or share data locally to conserve energy and power down unused or unneeded nodes. The AC Hierarchy [7, 8, 38] forms hierarchical clusters covering the entire sensor network and provides a high level programming abstraction for division or simplification of the sensor network. Finally, GLS [19] and MASH [33] provide cluster-based location or query services for locating data or nodes. Unlike these cluster-based or group-based algorithms we provide the abstraction of tracking groups linked directly to environmental events of interest. Although some of these algorithms such as GLS could be used in parallel with our work for object lookup, and although various ideas from these protocols are similar or could be used to enhance the efficiency or functionality of our modules, none provide sufficient support for entity formation around environmental events of interest and end-to-end connection establishment and maintenance between moving entities.

Our work is also complementary to several research efforts that aim to provide new abstractions and paradigms for distributed computing in sensor networks. For example, MagnetOS [2], exports the illusion of a single Java virtual machine on top of a distributed sensor network. The application programmer writes a single Java program. The runtime system is responsible for code partitioning, placement, and automatic migration such that total energy consumption is minimized. Mate [18] is another example of a virtual machine developed for sensor networks. It implements its own bytecode interpreter, built on top of TinyOS [11]. The interpreter provides high-level instructions (such as an atomic message send) which the machine can interpret and execute.

To the authors' knowledge, the communication architecture proposed in this paper is the first that provides middleware support to tracking applications for group formation around environmental events, end-to-end connection establishment between different entities, and abstract entity maintenance to ensure that a single entity is formed and maintained for every event in the environment. The architecture ensures a one to one relationship between abstract entities and environmental events thus simplifying communication. It reduces programming complexity by allowing communication with entities rather than individual nodes.

### 3 Service Architecture

The ability of a sensor network to closely interact with the environment in which it has been deployed gives rise to a multitude of applications in which code execution is tightly linked to the locations of environmental events. Appropriate communication abstractions are required to isolate distributed application programmers from accounting for the changing locations of environmental events in the vicinity of which the communication end-points of their applica-

tions are located. Our architecture provides such abstractions by a combination of (i) a team management framework for maintaining proper mapping between communication endpoints and external events (ii) a transport layer protocol for communication among event-related endpoints. Together, they maintain communication end-points associated with mobile external events, as well as maintain connectivity among such endpoints. This architecture is independent of the underlying radio, data link, and network layer protocols making it applicable in principle to an array of sensor network platforms.

Such programming abstractions are desired in applications where one wishes to interact with physical events in the environment that, for one reason or another, do not communicate directly with the network. By forming an abstract entity that moves with the event, we can associate state and behavior with the physical event. For a hostile target being tracked, this state and behavior could include monitoring the number of shots fired from a tank or the distance an object has traveled.

#### 3.1 The Entity Communication Problem

The problem addressed by our architecture is more formally described as follows. We consider a dynamically changing set  $S$  of events in the physical environment of the sensor network. Let the physical location of each event  $E_i \in S$  at time  $t$  be denoted  $L_i(t)$ . A node is said to be in the *vicinity* of event  $E_i$  at time  $t$  if it is within sensor range of the event's location,  $L_i(t)$ . In this paper, we assume that environmental events are localized. In other words, their location is described by a single point in space, as opposed to an area. This definition applies to tracking vehicles, finding survivors, monitoring wild animals, or detecting localized fires. It does not apply to applications involving distributed phenomena such as detection of large chemical spills. We assume that events can be detected independently by individual nodes in the sensor network based on their local measurements. For example, detecting a magnetic signature in a desert battle area would usually be indicative of a passing armored vehicle. Finally, we assume that events are sparse. In other words, the signatures of different targets are generally not overlapping.

Let  $T_i$  denote the set of nodes in the vicinity of event  $E_i$ . The objective of our architecture is to maintain a *unique* addressable destination associated with each event  $E_i$ , such that sending data to this logical event address causes delivery of this data to  $T_i$  regardless of the location  $L_i(t)$ . In the current implementation, we elect a leader out of set  $T_i$ . The leader, among other things, is responsible for communication with remote destinations. Hence, in the above problem statement, we define delivery of a message to  $T_i$  as delivery

of the message to the current entity leader who by definition belongs to the set  $T_i$ . What the leader does with the message is an orthogonal issue in our architecture.

Note that once the aforementioned addressing and communication problem is solved, it becomes trivial to associate multiple communication end-points with each entity simply by demultiplexing the received message based on a port number in the message header, in the same sense that UDP creates multiple ports over IP.

### 3.2 Sensor Network Assumptions

Our underlying sensor network typically consists of thousands of small sensor nodes thrown arbitrarily (e.g., from the air) onto a large target area, such as a battlefield or the scene of a natural disaster. Individual nodes have resource limitations associated with small physical size including low-power batteries, relatively slow processors, and limited memories. They are capable of wireless communication and once deployed form a large-scale ad hoc network. A key assumption is that the formed sensor network has no pre-existent infrastructure or centralized services. It is precisely the difficulty of creating such an infrastructure in harsh or inaccessible environments that motivates the sensor network approach. An example of computationally equipped wireless sensor devices that meet the above description is the MICA mote [12], which we use in our experimental prototype.

Once deployed, nodes in a sensor network are assumed to establish their location and remain motionless except due to environmental factors such as wind and water. In an important departure from the typical mobile ad hoc wireless network model, nodes in sensor network literature do not have IP-address, and do not run the TCP/IP protocol suite. Instead of possessing unique ID's, sensor network nodes are usually referenced by attributes such as location. Both localization services [4, 30] that establish sensor network coordinate frameworks, and location-based routing services [17, 3] that route messages geographically have been discussed at length in previous literature.

Sensor nodes may perform local processing as appropriate for their particular application. This could be aggregating and reporting raw data, triangulating the position of an event, coming to agreement about an actuation or reporting strategy, or performing distributed event analysis. A distributed application, such as a distributed intrusion response system, may need to pass the results of such local processing among the respective groups of sensors to coordinate a sensor network reaction.

Our service can be thought of as a distributed protocol that sits in the transport layer of the sensor node's protocol stack. In its basic form, the protocol implementation

consists of two modules, namely, the *entity management module* (EMM), and the *entity connection module* (ECM). These modules are shown in Figure 1. As the name suggests, the EMM forms a local entity in response to sensor readings at the locations of environmental events. It maintains the unique identity of this entity as the event of interest migrates in the environment. The ECM provides a means for entity registration, maintains communication end-points, and provides connectivity to allow communication among different entities. The following sections discuss the details of the APIs and implementations of the aforementioned modules.

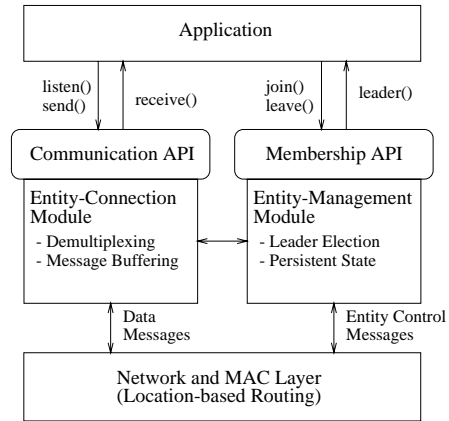


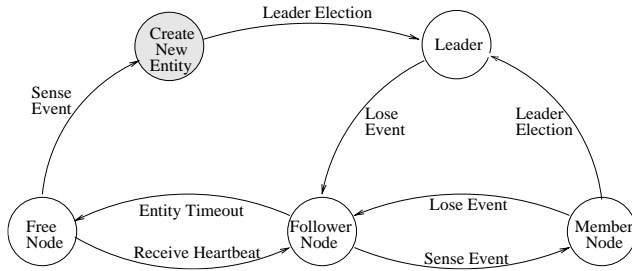
Figure 1. Service Architecture

### 3.3 Entity Management Module

The entity management module (EMM) provides an entity formation and maintenance service. The EMM has several essential functions. First, an entity must be created and identified when an event first occurs in the sensor network. Second, once established, the EMM must maintain this single entity and prevent spurious entities from forming around a previously abstracted event. While an entity exists the EMM must maintain its persistent state such as its unique identity which signifies the local environmental event. Finally, the EMM is responsible for ensuring that nodes that sense an event for the first time know when to become a member of an existing entity and when to spawn a new entity around the sensed event. These functions are described below.

#### 3.3.1 Functional Overview

At any given time, multiple events in the environment give rise to multiple entities which host the communication end-points in the sensor network. An entity is a set of nodes in the vicinity of a single environmental event. We call nodes whose sensors detect the signature of the corresponding event, *entity members*. These nodes are said to be within



**Figure 2. Node state transition diagram**

the *sensory horizon* of the target event. To the rest of the sensor network, an entity acts as a single whole. The fact that the entity may consist of multiple nodes is hidden and the identities of these nodes are abstracted away.

Entity members volunteer to be an *entity leader*, a central node responsible for communication and group maintenance, as well as running entity-specific application code. Once the entity leader no longer detects the event, it hands-off leadership (and current state) to another node by sending a *relinquish leadership* message. As the event migrates or expands, the leadership handoff mechanism built into the EMM ensures that the entity and its stored application state migrate with it. The application on a node is informed when the node becomes a leader so that it can pick up computation from where the previous leader left it. The application is also notified when the node ceases to be leader.

A key requirement is to ensure that only one entity is spawned for the same environmental event. The EMM protocol achieves this requirement by announcing the existence of an entity to nearby nodes within a distance called the *awareness horizon*. By design, the awareness horizon is larger than the sensory horizon. Nodes in the awareness horizon that cannot sense the event are called *entity followers*, as distinguished from entity members. These nodes are prevented from spawning new entities.

Nodes join the awareness horizon upon the reception of a bounded-hop broadcast (*heartbeat*) from the nearby EMM leader. Upon receiving a heartbeat, a node sets a corresponding *entity timeout timer*. The node ceases to be a follower when the timer expires. The timer is re-initialized upon the receipt of each new heartbeat. If the node senses the event signature before the entity timeout timer expires it becomes a member of the existing entity. If the timer had already expired the node is no longer a follower and will create a new entity. This mechanism prevents multiple entities from being spawned for the same environmental event. Figure 2 depicts the node state transition diagram between follower, member, and leader states, as well as the free state in which a node is not cognizant of any entities. For readability, the only transitions shown are those between different states (as opposed to loop-backs from a state to itself).

Note that, while inhibiting entity followers from creating new entities is essential for preventing redundant representation of the same target, it has one unfortunate side effect. Namely, the mechanism may prevent nodes from reporting secondary events in the vicinity of events already reported. Since we assume that events are sparse, the hope is that cases like the above are not common.

### 3.3.2 Entity Uniqueness

Entity uniqueness is the algorithm property that states that only one entity is associated with any given environmental event. In this section, we take a closer look at the conditions under which this property holds true. In general, there are two cases in which entity uniqueness can be compromised. The first case occurs at excessive target speeds. If the target moves in the environment fast enough, far apart nodes can detect it at about the same time and create independent entities to represent it. The second case occurs due to message loss or node failures which may prevent proper leadership handoff. Consequently a new leader may emerge that does not inherit the right entity identity from the old leader, causing a different entity to emerge for the same environmental event. In the following, we quantify the maximum event speed that preserves entity uniqueness and discuss provisions to ensure robustness in the face of failures.

**Event Speed:** The key rule which inhibits creation of duplicate entities is that followers of existing entities cannot spawn new entities. Instead, when they eventually sense the event, they must join the membership of the entity of which they were followers. By extending awareness of the event (i.e., the awareness horizon) beyond its sensory horizon we can ensure that new nodes will always become aware of the current entity before they sense the event. Hence, a single unique entity will exist for each event in the environment. The above uniqueness property is violated only if the event moves fast enough in the environment such that it is sensed by nodes outside of the awareness horizon before information of this event is propagated to them. Controlling the awareness horizon therefore determines the maximum tolerable event velocity as will be detailed below.

Note that a new leader is elected once the old one stops sensing the target. This new leader will cause the center of the awareness horizon to shift to its new location. If leader re-election and heartbeat propagation took zero time, the system could theoretically track infinitely fast targets as long as the awareness horizon was at least double the sensory horizon. This is because the current leader would always be within sensor radius from the target and no other node within the sensory horizon could be more than twice the sensor radius away from the leader. Hence, all nodes who sense the target are always within the awareness horizon and are therefore inhibited from creating new entities. In

reality, however, leader re-election and heartbeat propagation take time. If the maximum combined leader re-election and heartbeat propagation delay was  $D$ , it is easy to show that the maximum speed that preserves entity uniqueness is  $(\text{awareness horizon} - 2 \cdot \text{sensory horizon})/D$ . It should be noted that the above is a conservative estimate. Entity uniqueness will not be compromised immediately at higher target speeds.

**Robustness to Message Loss and Failure:** To prevent handoff failure in the case that an entity leader dies or otherwise fails to send out the relinquish heartbeat message, each entity member sets a *failed leader timer*. This timer, upon expiration, prompts an entity member to assume the entity leader role and begin sending heartbeats after an additional random delay (to prevent simultaneous takeover collisions). This failed leader timer must be set to a value larger than the heartbeat period, the interval between heartbeats, to ensure that timer expiration does not occur prematurely while the current leader is still alive. Depending on expected message loss, one might also set this timer to a value greater than two or three times the heartbeat period to prevent inopportune and premature handoff when heartbeats are lost or subject to collisions. Note the delay that a node waits before assuming the entity leader role could be determined in accordance with the strength of a node's sensor reading, whether or not this sensor reading is growing or shrinking in strength, the number of entity members that are direct neighbors of that node, or by some other appropriate metric.

Message loss can also prevent nodes within the awareness horizon from getting the leader's heartbeats. Consequently, these nodes may not become aware of the entity and may create a spurious one when they sense the event. To kill such spurious entities, we employ a mechanism that associates larger weights with older entities and biases nodes against joining entities with smaller weights. The mechanism maintains an *alive counter* at the leader of each entity. This counter is propagated through heartbeats and its value is accumulated across leader handoffs. When a new entity is first created, its counter is initialized to 0. This value is then incremented for each heartbeat sent out and is therefore a reflection of how long the entity has remained in the network. When a node tries to spawn a new entity, every neighbor that is already part of an entity with a higher alive counter ignores the new node. Hence, the faulty node is isolated. The mechanism will send a kill message to the faulty node to request termination of its spurious entity.

The above mechanism serves to prevent spurious groups from forming in the presence of message loss, but fails to handle the case where events of the same signature migrate across one another's path. To handle this more complex scenario we define a compile time specified threshold, *min time alive*, to ensure entities that have existed over some time period remain after crossing paths with an even older

entity. When a node of entity  $E1$  receives a heartbeat from the leader of another entity  $E2$  and both entities have an alive counter set greater than the min time alive threshold, we require that both entities coexist. In this case, nodes independently apply the EMM protocol with respect to each entity. They may be within the awareness horizon of multiple entities at the same time. When they sense the event, they become members of all entities that exceed the min time alive threshold of which they are aware.

### 3.4 Entity Management API

The API exported by the EMM consists of *join(signature)* and *leave(signature)* primitives which the application calls when it first senses or stops sensing an event signature respectively. The code of the signature is passed as the input parameter to these primitives. For example, if the magnetic signature of a vehicle is sensed, the node calls *join(Magnetic)*. Unlike traditional group communication, the *join()* does not take a group identifier as input. Instead, it returns as output the identity of the environmental event the application just sensed (i.e., the identity of the entity for which the node was a follower at the time *join()* was called). If the node is a follower of multiple nodes, a list of identities is returned. The semantics are that the node has joined the respective list of entities. If a node is not a follower of any entity, a new entity is created when *join()* is called and the code of the new entity is returned.

The EMM also requires the application to implement a handler for an upcall called *leader(entity,on\_off)*. The upcall contains an entity id as a parameter as well as a boolean that tells the application that its node has just become or ceased to be leader of the named entity. Application code would typically check whether it is the leader or a member, and execute the corresponding part of its typically distributed data processing algorithm based on the assigned role.

The last part of the API is a *store(entity,state)* and *get(entity,state)* call that allows the application on the leader node to save and retrieve persistent state of a named entity. The entity name is passed as input parameter to the call. Typically, the leader would save its state after each iteration. This state is transmitted in the EMM heartbeats to all members of the entity. Upon leader handoff, the new leader would use the above API to get the most recent previously communicated state. It would then resume the iterative application from that point onwards.

### 3.5 Entity Connection Module

The entity connection module (ECM) provides a basic end-to-end location and communication service between mobile entities. ECM is therefore the equivalent of UDP for sensor

networks, with the exception that destinations are migratory entity leaders, not IP hosts. An application can utilize the ECM’s API to communicate messages to and from logical entities without concern for where that entity resides, how it is maintained, or what particular nodes compose it. An application programmer can therefore initiate queries and interact with environmental events that migrate throughout the sensor network.

The ECM exports a subset of a socket-like API. All applications are assumed to have well-known ports. In the current protocol 256 ports are supported based on a byte in the message header. We do not support dynamic application-to-port binding. This is because in our target platform, namely, Berkeley’s MICA motes running TinyOS, applications are structured as a graph of permanently wired modules. The ECM demultiplexes incoming messages as upcalls to different application modules depending on their port number. The association of port numbers and upper layer modules is defined in a compile-time configuration. At run-time an application can call *listen()* to notify the ECM that it is ready to receive messages on its assigned port. Subsequently, the ECM propagates messages on this port to the application. If a message arrives for a port on which no application is listening, the message is dropped.

Connections are identified by a  $\langle \textit{Entity ID}, \textit{Port Num} \rangle$  pair. When an entity is spawned, entity registration is invoked by the ECM. This registration utilizes a directory service similar to the indirection infrastructure described in [35]. Namely, each entity maintains replicated pointers to its current location in a region of the sensor network determined by a hash function. The hash key is the signature identifier associated with the entity. By hashing this key, the ECM can determine the location of the directory region associated with a particular type of environmental event, then query the directory for all entities that are currently following events of that type. Queries to this directory service supply entity leader information pertinent to establishing mobile connections.

When connecting with an entity, the ECM looks up the last known entity leader based on the  $\langle \textit{Entity ID}, \textit{Port Num} \rangle$  pair provided in the application call. If this information is older than a specified threshold the directory service is contacted for updated information. The returned last-known leader is used as a connection point for communication. Upon receiving a message, an endpoint updates its table of last-known leaders with that contained in the header. The more traffic exchanged between the endpoints, the more up-to-date the leader information is.

Leadership information is retained in the ECM in a limited-size table. When the table is full, replacement is done on a least-recently-used basis. The ECM of an entity periodically refreshes the directory region, at an interval called the

*directory refresh rate* to ensure that its information remains up to date. In addition, past followers of an entity remember the location of the last known leader for a time interval that exceeds the directory refresh rate. Hence, messages sent to the old location of an entity are forwarded to the current location when they intercept the entity’s trail.

## 4 Simulation

To fully understand and validate our proposed architecture, we implement our design in GloMoSim V2.03, a popular wireless network simulator. GloMoSim was developed as a modular library of components that contribute to an extensible, robust, and dynamic simulation of wireless networks. By isolating nodes’ communication layers into independent modules, GloMoSim allows researchers to “plug and play” different protocols (i.e. protocols that they develop and implement) without concern for the inner workings of other architectural layers. The simulation environment allows us to apply our modules to the problem of event tracking. Specifically, by simulating event tracking in GloMoSim using our EMM and ECM modules, we are able to analyze the effects of architectural parameters on performance and understand how our architecture can be tuned to solve real world tracking problems.

### 4.1 Scenario

We modified the Transport and Application layers of GloMoSim to simulate a hypothetical sensor network environment consisting of nodes communicating over a 220 meter radius, which is typical for sensor nodes. The lower level protocols of our wireless network include Geographic Forwarding [16] over IP in the Network layer and 802.11B in the MAC layer. We simulated 32 byte packets sent in a 200 kbps wireless medium, a slightly larger bandwidth than the capabilities of today’s sensor devices (50kbps-100kbps) to account for future improvements. At the physical layer, we use a Two-Ray Pathloss model with SNR-Bounded Noise at the receiving node. The model allows noise, attenuation, and subsequent loss on the wireless channel to be simulated, as opposed to perfect reception within a hypothetical radius.

Sensor nodes are equipped with sensors that poll their environment for specific events (e.g., acoustic sensors that monitor and can recognize certain acoustic signatures such as tank movement). In our experiments, a number of nodes are uniformly distributed in a 1,400 x 1,400 meter field. To test the ability of our architecture to track a moving target, we simulate an object moving across the field in a straight line. The moving object is tracked (presumably using acoustic or magnetic sensors) with a sensor polling period of 0.05 seconds, a granularity high enough to ensure up to date read-

ings. Sensors register a target up to a sensing radius of approximately 100 meters.

For our tests we employ a simple application that computes an event’s position through entity member reports to the entity leader. Entity members poll their sensors and send periodic updates to the corresponding entity leader notifying this leader of their current sensor reading and position. The leader computes the weighted average of the position of reporting entity members. The weighting is by sensor reading since higher readings presumably mean a closer target. This average value is sent back every 0.5 seconds to a “friendly-force” entity at a static location in the network. Upon receiving a report, this entity responds with a confirmation message sent back to the reporting entity. This feature allows us to test our architecture’s ability to maintain end-to-end connectivity and forwarding between entities.

## 4.2 Simulation Results

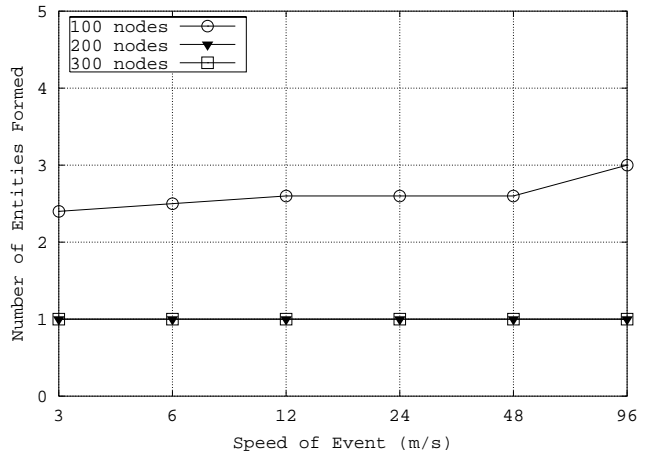
The objective of our simulation is to understand the effect of algorithm parameters on tracking, as well as estimate costs such as the energy consumed. In all experiments an entity is spawned and migrates with the moving target. Entity uniqueness should be maintained for a run to be successful. Hence, we count the number of entities that form around the moving target during the course of a simulation to determine whether or not our architecture was successful in establishing and maintaining a single entity per event. In energy cost experiments, we compute the energy consumed during send and receive operations in accordance with transmit and receive currents of the MICA notes [23]. CPU energy consumed constitutes a constant overhead.

Each point in the graphs below represents the average of 10 runs to ensure a statistical significance at the 0.05 level. In the subsequent analysis when we claim a target is *trackable* at a specified speed we mean that for all 10 trials, a single entity was formed and tracked during each trial. The two key parameters of the algorithm, whose settings determine performance are the EMM leader heartbeat period and the awareness horizon. This leader heartbeat period defines how often the entity leader sends heartbeats to members (and followers). The awareness horizon, in our experiments, defines how many hops the heartbeats are propagated. Other algorithm parameters are automatically computed depending on the settings of the above two. Namely, we require that the failed leader timer (used to detect leader failure) be set to a value twice greater than the heartbeat period to ensure that no member takes over leadership while a current leader is still sending heartbeats. Similarly we require that the entity timeout period (used to free follower nodes) be approximately 1.5 times the failed leader period to ensure that no follower leaves the group before an en-

tity member could properly take over leadership and begin sending heartbeats.

Below, we present and discuss those parameters that we feel are most influential to the problem addressed and the solution presented. We initially start with those parameters that are determined by the network or otherwise outside of the designer’s control. We then analyze parameters that can be set by the designer. For these graphs we choose to display the heartbeat timer on the x-axis to analyze its affect on chosen metrics. We vary the range of heartbeat values from graph to graph to demonstrate trends in the timer and show what we feel is the most relevant and interesting information for understanding our architecture’s performance.

### 4.2.1 Setting Node Density



**Figure 3. Effect of node density on number of groups formed (heartbeat period = 375 ms, awareness horizon = 1 hop)**

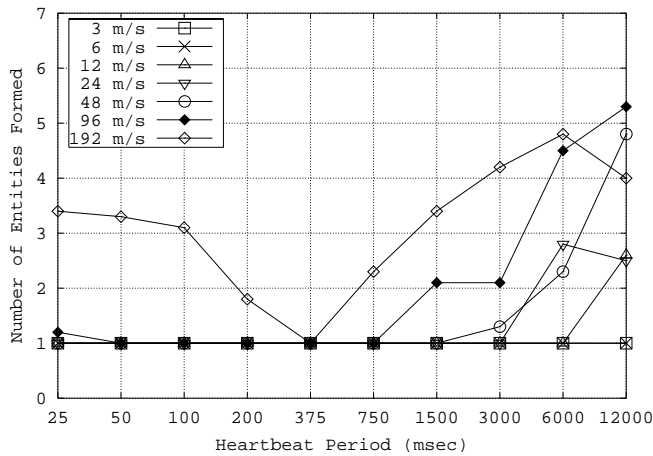
It is often assumed in sensor network research that the node density is high enough to ensure that all nodes are within communication range of several other nodes at all times. We begin by understanding the effect of node density on the performance of our algorithm. In this experiment, we vary the number of nodes in the rectangular field, and observe the number of formed entities around the moving target. The experiment is repeated for different target speeds. A run is successful if only one entity is formed. In the experiments below, we choose the awareness horizon to be one hop, which as we show later, represents a worst case from the perspective of trackability at the target speeds considered.

Figure 3 demonstrates the results. We see that independent of event speed, our architecture is capable of maintaining the formed logical entity when the number of nodes is 200

or higher which corresponds to an average distance of about 140 meters between any two nodes. Thus, in the rest of this section we fixed the number of sensor nodes to 200, as a rough estimate of “sufficient” node density.

#### 4.2.2 Effect of Leadership Handoff

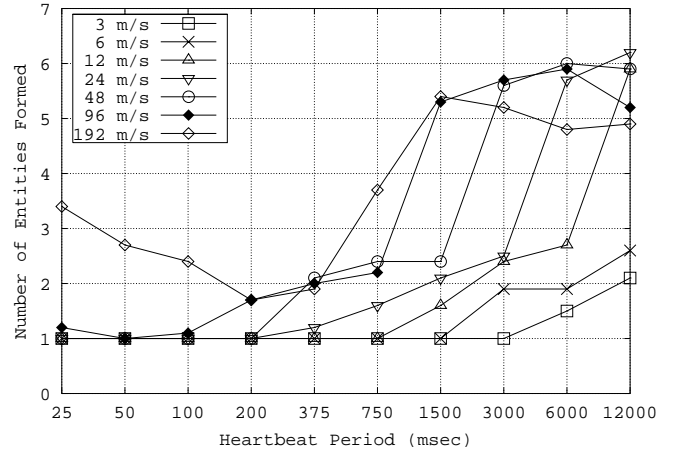
Next, we explore the effect of the leadership handoff mechanism on performance. While our architecture includes a leadership handoff mechanism that explicitly notifies entity members when a new leader should be elected, we compare that mechanism to the case where the old leader simply dies in which case a timeout must elapse before the new leader election starts. Figures 4 and 5 compare the average number of entities formed around the target for different target speeds with and without the handoff mechanism respectively. For each target speed, we vary the leader heartbeat period. Larger periods mean that more time will elapse before leader failure is noticed. Observe that the curves in Figure 4 are generally lower than in Figure 5, indicating a larger fraction of successful runs. Remember that each point on each curve is the average of 10 runs. Points indicating a single formed entity mean that all 10 runs were successful. The handoff mechanism is more successful in tracking since it avoids the extra delay in leadership handoff making it less likely that the target will move to where it can be sensed outside of the awareness horizon, thus causing a spurious entity to emerge. Thus, in the rest of our analysis we only consider simulations where our handoff mechanism is present.



**Figure 4. Groups formed with our explicit handoff mechanism (awareness horizon = 1 hop)**

In addition, it is interesting how the choice of the heartbeat period strongly influences our architecture’s ability to track

an event. Slow periods will result in a slower transition resulting in the event migrating beyond the awareness horizon. Fast periods result in a congested channel which increases message loss and prevents nodes from hearing about an approaching event. In between these extremes, an optimal choice of the heartbeat period can be made. This choice will be investigated in more detail later in the section.

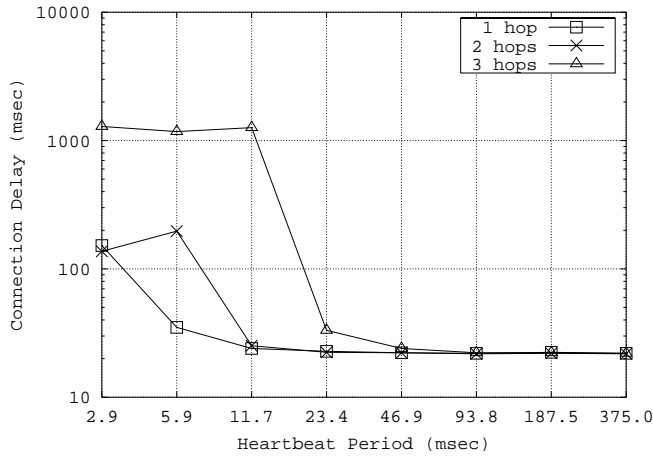


**Figure 5. Groups formed without our explicit handoff mechanism (awareness horizon = 1 hop)**

#### 4.2.3 Limits on Heartbeat Period

Next, we analyze the overhead of the algorithm by exploring the point at which it saturates the underlying network. This saturation point depends on the setting of the heartbeat period and the awareness horizon. It is fairly obvious that decreasing the heartbeat period results in more frequent communication between the nodes and therefore the ability to track faster targets. This is based simply on the speed of response necessary for faster targets. However, the bandwidth limitations in the wireless medium place limits on our timer settings and constrain our architecture’s ability to track migrating events. To determine the bandwidth needs of our algorithm, we start with a very small leader heartbeat period (that saturates the network), then increase it gradually. We plot the resulting connection delay, which is the time it takes to send a message from the moving entity to the friendly-force entity. As the heartbeat period increases to the point when the network is no longer saturated with heartbeat traffic, we observe a sharp decrease in the connection delay. Figure 6 shows this effect. The experiment is repeated for different awareness horizons, expressed in the number of hops that leader heartbeats are propagated to. It is seen that when the horizon is increased, the onset of over-

load occurs earlier as more messages are communicated.



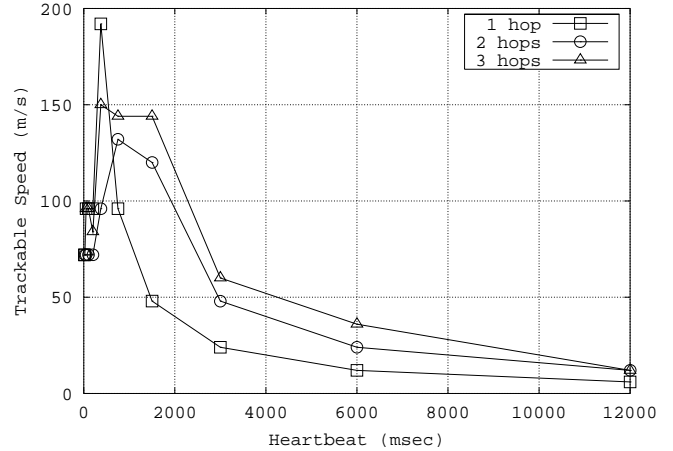
**Figure 6. Effect of timer settings on message delay (speed = 12 m/s)**

From Figure 6 we can see that the bandwidth of the wireless medium is fully saturated when the leader heartbeat period is reduced to approximately 2.9, 5.9, and 11.7 ms for an awareness horizon of = 1, 2, and 3 hops respectively. To conservatively avoid this saturation point and ensure enough bandwidth is left for alternate local traffic, we multiply these numbers by 5 (i.e., limit the worst case overhead of tracking to 20%). Hence, in the rest of the evaluation section, we consider only those leader heartbeat periods that are above 12.5, 25, and 50 ms for 1, 2, and 3 hops respectively. We next turn our attention to the selection of the leader heartbeat period and the awareness horizon, the two key parameters of the algorithm, subject to the above constraints.

#### 4.2.4 Heartbeat Period and Awareness Horizon

Our group management protocol works by propagating leader heartbeats a specified number of hops from the leader, which determines the awareness horizon. Increasing the hop count allows us to track faster events since it extends the area in which nodes know about the oncoming event and requires a faster event to migrate faster than its corresponding logical entity. However as previously mentioned, the awareness horizon restricts the maximum leader heartbeat period, another important parameter of the algorithm. In the following experiment, we study the coupling between these parameters to analyze their affect on the maximum trackable event speed. Specifically, we vary the leader heartbeat period over a range above the pre-determined saturation points, and compute for each period the maximum event speed at which tracking always succeeds in maintain-

ing a single entity for the moving target. A different curve is plotted for different awareness horizons.



**Figure 7. Trackable event speeds for varied heartbeat period:awareness horizon settings**

Figure 7 shows the results of the above experiment. Following the curves from right to left, we can see that up to a threshold, reducing the leader heartbeat period results in the ability to track faster events. However, at this threshold, network collisions, congestion, and message loss become dominant. Hence, the maximum trackable speed deteriorates. This is apparent from the sudden drop in the maximum trackable speed seen on the far left side of the graph (about 100 ms). We also note that increasing the awareness horizon increases the trackable speed for slower heartbeats, but also congests the channel faster resulting in an inability to track faster targets. In the graph we notice that the single hop awareness horizon remains below the 2 and 3 hop settings up until about 400 ms, the point at which all but the 1 hop settings begin to break down.

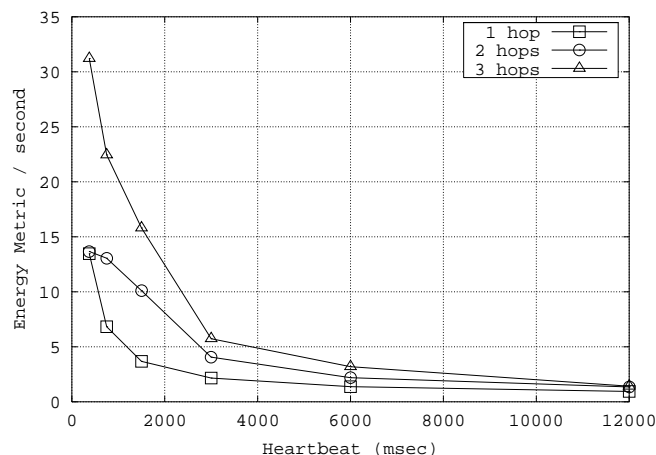
A designer's decision to set the heartbeat period and the awareness horizon should be guided by the limitations on trackable speed shown in Figure 7. The trends seen in this figure illustrate the fundamental trade-offs involved in parameter selection in our algorithm. While the axes can be scaled depending on other platform settings such as node spacing, the figure provides interesting insights into the choice of protocol settings. For example, the figure shows that for speeds lower than 120 m/s (i.e., slightly more than half our communication radius per second) a designer has a choice of heartbeat period and awareness horizon settings that jointly allow a given speed to be tracked. The candidate settings are those that result from intersecting a horizontal line (drawn at the desired speed) with each of the three plots representing the different awareness horizons. Each intersection point gives the heartbeat period needed

for the corresponding awareness horizon. In general, choosing a larger awareness horizon implies a more relaxed (i.e., larger) heartbeat period. For example, to track a target of speed 50 m/s, one can use a heartbeat period of 1600, 2900, and 4200 ms for a horizon of 1, 2, and 3 hops respectively. Interestingly, observe from Figure 6 that at those periods the network is fairly underloaded. When the target speed is higher than 120 m/s, however, larger values of awareness horizon fail, since they present a higher percentage of message loss and larger delays which allow spurious entities to be created. At those speeds a smaller awareness horizon is necessary.

If the designer has multiple choices, an important factor in choosing a particular combination of parameters is power consumption. The effect of parameter settings on power consumption is explored next.

#### 4.2.5 Energy Consumption

Aside from their influence on the maximum trackable speed, the leader heartbeat period and the awareness horizon also significantly influence message related energy consumption. For different hardware this effect will vary. However, the fundamental trends remain consistent.



**Figure 8. Energy consumption for varied heartbeat period:awareness horizon pairs**

Figure 8 shows energy consumption for varying the leader heartbeat period and awareness horizons explored in Figure 7. The effect of the heartbeat period on energy consumption is significant. An interesting point to notice, however, is that the alternative (horizon, period) tuples that track a particular speed consume almost the same amount of energy. Following up on the example from the previous section, the three alternative parameter tuples that track a speed of 50 m/s, namely, (1 hop, 1600 ms), (2 hops, 2900 ms), and

(3 hops, 4200 ms), consume roughly the same energy of 4 units. This is seen by finding the intersections of the vertical lines at 1600 ms, 2900 ms and 4200 ms with the energy curves for the corresponding horizon. It therefore appears that choosing a larger horizon does not have an advantage as long as the heartbeat period is appropriately chosen. This is intuitive. Increasing the awareness horizon allows using a larger heartbeat period. However, it also increases the number of messages sent, since each heartbeat is flooded in a larger radius. The net sum of messages exchanged, therefore, remains largely unaffected.

In view of the above, an argument can be made for smaller horizons, since they reduce the number of nodes inhibited from creating new entities, thus making the sensor network more responsive to the advent of new events into the environment. Nodes in a network with a large awareness horizon will attribute measurements of such new events to the entities they are already aware of; an effect that should be avoided.

The set of experiments presented above illustrate the ability of our architecture to track events at varied speeds in a sufficiently dense sensor network. A system designer can employ this architecture, choosing the leader heartbeat period and the awareness horizon in accordance with the expected speed of migrating events, available bandwidth of the wireless medium, and energy restrictions (required system lifetime) of the hardware being deployed. An implementation of our architecture on the MICA motes which is discussed in the following section.

## 5 Motes Implementation

We implemented our work on the MICA motes developed by X-Bow. Under our architecture we implemented a simple Geographic Forwarding [16] mechanism as well as a simple buffering MAC protocol that follows a send when ready strategy with no contention resolution. See [11] for more information on the MICA platform and TinyOS Operating System.

Our MICA test bed consists of 24 motes laid out in a 12 X 2 grid with motes placed one foot apart. Nodes communicate to neighboring nodes within a specified neighborhood grid size (NGS). The awareness horizon is set to 1 hop in accordance with the small size of the network. For our experiments we tested NGS sizes of 2 and 4.

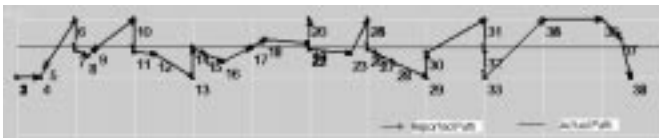
Nodes are employed with light sensors capable of detecting a shadow cast by the tracked event. Our goal was to track a rectangular object 1 square grid in size, moving at a speed of 1 grid per 10 seconds (GrPS). Nodes are programmed with our tracking architecture and an application which performs target location estimation. Additionally, nodes are

Neighborhood	1/5 grid/s	1/10 grid/s	1/15 grid/s
2	5	4	1
4	3	1	1

**Table 1. Entities formed over varied event speeds on the MICA testbed**

pre-configured to report a detected event to a fixed location in the sensor network (node 0) every time the aggregate event location changes.

Based on the limited bandwidth, energy restrictions, expected target speed, and radio radius of the MICA motes, we experimented with and chose our heartbeat, failed leader, and entity timeout timers to be 2, 5, and 8 seconds respectively.



**Figure 9. Reported location of tracked entity on MICA motes (Speed = 1/10 GrPS, NGS=4)**

Initial experimental data show that for speeds of 1/10, and 1/15 GrPS and a NGS of 4 grids, our architecture as deployed was capable of correctly forming, maintaining, and reporting the location of a group around the event of interest. The reported path of the event for one such trial is shown in Figure 9. In this experiment, the tracked vehicle was traveling in a horizontal line from left to right as shown by the horizontal axis in the figure. The jagged quality of the reported path is a result of the limited number of motes detecting and reporting a position for the target at any specific point in time. It is possible that when the communication radius and the speed of events vary, multiple entities can be formed. Table 1 shows the number of entities formed over different speeds in the MICA testbed. The creation of spurious entities at faster speeds is due largely to the use of an unreliable MAC layer in the experiments. Message loss, as mentioned in the paper, allows spurious entities to be created. The noisy nature of the measured track is due to the fact that the light sensors do not provide a measure of distance from the target. Hence, the best one can do is simply average the positions of all sensors detecting the event. In contrast, other sensors (such as magnetic sensors) can provide a better estimate of how far a measured target is. Triangulation will therefore result in a much more accurately estimated position.

Observe that the purpose of our experimental measurements

is to qualitatively illustrate the success of our scheme in practice. It is not our purpose to compare experimental measurements to simulation. We have purposely decided to use a standard wireless MAC layer in the simulator instead of the simplified unreliable custom MAC layer implemented on the actual motes. Consequently, different performance results are expected. We believe that the implementation of the motes MAC layer will mature significantly in the future, making it less interesting to seek quantitative performance statements on the current testbed at this time.

## 6 Conclusions and Future Work

In this work we provide a transport layer solution to entity maintenance and connectivity in sensor networks. Our proposed middleware service provides a novel way for programmers to relate to events in the environment without concern for topological and communication layer details irrelevant to their application. We establish entities in a one-to-one relationship with events to ensure correct and well defined behavior. Entities form and register with interested parties to allow unique identification and communication without regard for an event's location.

The analysis of our architecture demonstrates the effect of both uncontrollable (environment specific) and controllable (architecture specific) parameters on entity formation and maintenance. In accordance with an ideal sensor network, we require that at any time at least one node is capable of sensing the event being tracked. Under this assumption we demonstrate our architecture's capabilities and limitations in tracking events of varied speeds as a function of the pre-specified heartbeat and awareness horizon parameters. For a large field of relatively dense nodes, we show in simulation that our architecture is capable of tracking events that travel over half of the communication radius of a node per second (see Figure 7). Under optimal conditions in our simulation study, our architecture was able to track objects moving at about 88% of the communication radius. We additionally discuss the required settings for tracking events of varied speeds and the tradeoff of increasing the trackable speed and thereby increasing the amount of energy consumed.

In addition to simulation analysis, we provide an implementation of our architecture on the MICA test bed in our lab. In the presence of true fading and message loss we demonstrate the feasibility of our work for a simple application. Implementation results show the importance of a reliable MAC layer. Unlike the simulation, which used 802.11 to access the medium, the MAC layer in the implemented prototype used unreliable transmission. Consequently, the maximum tracked speed was significantly lower. The authors are currently implementing a reliable MAC layer for the motes platform. While our architecture provides the re-

quired mechanisms to create, maintain, and communicate with abstract entities in an environment, we have only begun to explore the possibilities of such sensor network related services. The applications and opportunities for sensor networks remain vast and mostly unexplored. This paper is a step towards a comprehensive coverage of research issues motivated by tracking problems in sensor networks.

## Acknowledgements

This work was supported in part by DARPA grant F33615-01-C-1905, MURI grant N00014-01-1-0576, and NSF grant CCR-0208769

## References

- [1] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan and J. Lilley, "The design and implementation of an intentional naming system," *Proceedings of 17th ACM SOSIP*, Kiawah Island, SC, Dec. 1999.
- [2] R. Barr, J. Bicket, D. Dantas, B. Du, T. Kim, B. Zhou and E. Sirer, "On the Need for System-Level Support for Ad hoc and Sensor Networks," *Operating Systems Review, ACM*, 36(2):1-5, April 2002
- [3] S. Basagni, I. Chlamtac, V. Syrotiuk and B. A. Woodward, "A Distance Routing Effect Algorithm for Mobility (DREAM)," *ACM/IEEE Mobicom 1998*, pp. 76-84, Dallas, TX, October 1998.
- [4] N. Bulusu, J. Heidemann and D. Estrin, "GPS-less Low Cost Outdoor Localization For Very Small Devices," *IEEE Personal Communications, Special Issue on "Smart Spaces and Environments"*, 7(5):28-34, October 2000.
- [5] A. Cerpa and D. Estrin, "ASCENT: Adaptive Self-Configuring Sensor Networks Topologies," *IEEE Infocom 2002*, New York, NY, USA, June 23-23, 2002.
- [6] B. Chen, K. Jamieson, H. Balakrishnan and R. Morris, "SPAN: An energy efficient coordination algorithm for topology maintenance in ad hoc wireless networks," *ACM/IEEE Mobicom 2001*, pp. 85-96, Rome, Italy, July 2001.
- [7] D. Coore, R Nagpal and R Weiss, "Paradigms for Structure in an Amorphous Computer," AI Memo 1614, MIT, October, 1997.
- [8] D. Estrin, R. Govindan, J. Heidemann and S. Kumar, "Next Century Challenges: Scalable Coordination in Sensor Networks," *ACM/IEEE Mobicom 1999*, pp. 263-270, Seattle, Washington, August 1999.
- [9] Z. J. Haas and M. Pearlman and Prince Samar, "The Zone Routing Protocol (ZRP) for Ad-Hoc Networks," Internet Draft, July 2002.
- [10] W. Heinzelman, A. Chandrakasan and H. Balakrishnan, "Energy-efficient communication protocol for wireless sensor networks," *Proc. Hawaii Intl. Conf. System Sciences*, pp. 3005-3014, Hawaii, 2000.
- [11] J. Hill, R. Szewczyk, A. Wood, S. Hollar, D. Culler and K. Pister, "System Architecture Directions for Networked Sensors," *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, Cambridge, M.A., November 2000.
- [12] M. Horton, D. Culler, K. Pister, J. Hill, R. Szewczyk and A. Woo. "Mica: The commercialization of microsensor motes," *Sensors Online*, 19(4), April 2002. <http://www.sensormag.com/articles/0402/index.htm>.
- [13] C. Intanagonwiwat, R. Govindan and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks," *ACM/IEEE Mobicom 2000*, Boston, MA, August 2000.
- [14] D. Johnson and D. Maltz "Dynamic Source Routing (DSR) in Ad Hoc Wireless Networks," *Mobile Computing* pp. 153-181, 1996.
- [15] F. Kamoun and L. Kleinrock, "Hierarchical Routing for Large Networks: Performance Evaluation and Optimization," *Computer Networks 1977(1)*, pp. 155-174.
- [16] B. Karp and H. T. Kung, "Greedy perimeter stateless routing (GPSR) for wireless networks," *ACM/IEEE Mobicom 2000*, pp. 243-254, Boston, MA, August 2000.
- [17] Young-Bae Ko and Nitin H. Vaidya, "Location-Aided Routing(LAR) in Mobile Ad Hoc Networks," *ACM/IEEE Mobicom 1998*, Dallas, TX, October 1998.
- [18] P. Levis and D. Culler "Mate? - a Virtual Machine for Tiny Networked Sensors," *ASPLOS*, Dec. 2002
- [19] J. Li, J. Jannotti, D. S. J. De Couto, D. R. Karger, and R. Morris, "A Scalable Location Service for Geographic Ad hoc Routing," *ACM/IEEE Mobicom 2000*, pp. 120-130, Boston, MA, August 2000.
- [20] C. R. Lin and M. Gerla, "Adaptive Clustering for Mobile Wireless Networks," *IEEE Journal on Selected Areas in Communication 1997*, 15(7).

- [21] A. Mainwaring, J. Polastre, R. Szewczyk and D. Culler "Sensor Networks for Habitat Monitoring," *2002 ACM International Workshop on Wireless Sensor Networks and Applications*, Sept. 2002.
- [22] A. B. McDonald and T Znati, "A Mobility Based Framework for Adaptive Clustering in Wireless Ad-Hoc Networks," *IEEE Journal On Selected Areas in Communication*, 17(8), August 1999.
- [23] MICA: Wireless Measurement System. URL: [http://www.xbow.com/Products/Product\\_pdf\\_files/](http://www.xbow.com/Products/Product_pdf_files/), Crossbow Technology Inc., San Jose CA.
- [24] R. Nagpal and D. Coore, "An Algorithm for Group Formation in an Amorphous Computer," *Proc. 10th International Conference on Parallel and Distributed Computing Systems (PDCS'98)*, Nevada, Oct 1998.
- [25] V. Park and M.S. Corson, "A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks," *IEEE Infocom 1997*, April 1997.
- [26] C. Perkins, Ed., "IP Mobility Support," RFC 2002, IETF, Oct. 1996.
- [27] C. Perkins and P. Bhagwat, "Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers," *ACM Sigcomm 1994*, pp. 212-225, London, UK, Sept. 1994.
- [28] C. Perkins and D. Johnson, "Route Optimization in Mobile IP," Internet Draft, September 2001.
- [29] C. Perkins and E. Royer, "Ad-hoc On Demand Distance Vector Routing," *2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'99)*, New Orleans, Louisiana, February 1999.
- [30] Radihika Nagpal, "Organizing a Global Coordinate System from Local Information on an Amorphous Computer," AI Memo 1666, MIT, August 1999.
- [31] R. Ramanathan and M. Steenstrup, "Hierarchically-organized, multipop mobile wireless networks for quality of service support," *Mobile Networks and Applications 1998*.
- [32] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker, "A Scalable Contentaddressable network," *ACM Sigcomm 2001*, San Diego, CA, Aug. 2001.
- [33] A. Rosenstein, J. Li and S. Y. Tong "The Multicasting Archie Server Hierarchy," Project Home-Page. URL: <http://www.cs.ucla.edu/adam/mash.html>.
- [34] A.C. Snoeren and H. Balakrishnan, "An end-to-end approach to host mobility," *ACM/IEEE Mobicom 2000*, Boston, MA, August 2000.
- [35] Ion Stoica, Daniel Adkins, Shelley Zhaung, Scott Shenker, and Sonesh Surana, "Internet Indirection Infrastructure," *ACM Sigcomm 2002*, pp. 73-86, Pittsburgh, PA, August 2002.
- [36] I. Stoica, R. Morris, D Liben-Nowell, D. Karger, M.Kaashoek, F. DeBek and H. Balakrishnan. "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications," *To Appear in ACM/IEEE Transactions on Networking*
- [37] L. Subramanian and R. H. Katz, "An Architecture for Building Self-Configurable Systems," *ACM/IEEE Workshop on Mobile Ad Hoc Networking and Computing*, August 2000.
- [38] D. G. Thaler and C. V. Ravishankar, "Distributed top-down hierarchy construction," *IEEE Infocom 1998*, San Francisco, CA, March 1998.
- [39] P. F. Tsuchiya, "The Landmark Hierarchy: A New Hierarchy for Routing in Very Large Networks," *ACM Sigcomm 1988*, Stanford, CA, August 1988.
- [40] N.H. Vaidya P. Krishna, M. Chatterjee and D.K. Pradhan, "A Cluster-Based Approach for Routing in Dynamic Networks," *ACM Computer Communications Review*, 27(2), April 1997.
- [41] A. Woo and D. Culler, "A Transmission Control Scheme for Media Access in Sensor Networks," *ACM/IEEE Mobicom 2001*, Rome, Italy, July 2001.
- [42] A. Wood and J. Stankovic. "Denial of Service in sensor networks" *IEEE Computer*, 35(10), October 2002.
- [43] Y. Xu, J. Heidemann and D. Estrin, "Geographic-informed Energy Conservation for Ad Hoc Routing," *ACM/IEEE Mobicom 2001*, Rome, Italy, July 2001.
- [44] F. Ye, H. Luo, J. Cheng, S. Lu and L. Zhang, "A Two-tier Data Dissemination Model for Large-scale Wireless Sensor Networks," *ACM/IEEE Mobicom 2002*, Atlanta, Georgia, Sep. 2002.
- [45] W. Ye, J. Heidemann, D. Estrin, "An Energy-Efficient MAC Protocol for Wireless Sensor Networks," *IEEE Infocom 2002*, New York, USA, June 2002.