

# Distributed Event Tracking and Transport in Sensor Networks

Anthony Wood, Brian Blum, Prashant Nagaraddi  
Department of Computer Science  
University of Virginia, Charlottesville, VA 22904  
{*wood, bblum, pnn7f*}@*virginia.edu*

May 3, 2002

## Abstract

Smart sensor networks require lightweight, distributed protocols and algorithms to achieve their goals while working within the tightly constrained limitations of their hardware and software. We consider networks of relatively fixed nodes that are interested in forming teams that track mobile events through the network, collaborating in some application-specific manner.

We present a suite of modules to provide team formation, management, and communication in a sensor network. The Team/Leader Management (TLM) module recruits team members around an event and manages leadership transitions. The Tracking Transport Protocol (TTP) provides basic end-to-end transport service between the distributed teams. Both use a generic and replaceable Group Management (GM) module to provide basic sensor-network appropriate group services.

To evaluate our mechanism, we simulate a simple event position tracking application. Our results show that the TLM preserves team continuity for sensor ranges matched to network densities, and that the TTP enables reporting to a remote endpoint.

## 1 Introduction

Sensor networks possess qualities that make traditional approaches to distributed computing inappropriate. Individual nodes have resource limitations associated with small physical size, including low-power batteries, relatively slow processors, and limited memories. They are usually connected via simple wireless links.

Nodes combine to form a large-scale ad-hoc network.

The network usually has no pre-existent infrastructure or centralized services, and may be deployed in harsh or inaccessible environments. It must be resilient to node failure, which is more common in a large-scale deployment.

Addresses used by the applications in such networks may be based on location instead of individual node identifiers. No individual node is important enough to be relied upon for crucial services. Special beaconing nodes deployed with the network can provide location services for the others to determine their own locations.

Nodes become important only as they cooperate with neighbors to achieve a network goal, which may require varying degrees of team formation, coordination, or consensus. Classic algorithms for distributed coordination are impractical because of burdensome communication, inappropriate fault models, and mobility. Lightweight protocols with looser semantics are needed.

Mobility can be manifested in a few distinct ways in a large-scale sensor network. The sensors may be able to move under their own power and direction. Localized density control, intruder pursuit, or environment manipulation could be applications for self-mobilizing nodes. Power requirements in these scenarios dwarf even the relatively high cost of radio transmission on small sensor nodes. We do not consider these applications to be centrally within the sensor network realm.

Sensor nodes may be attached to objects which move through the environment. Animal or vehicle tracking, optional components for PDAs, and wearable devices are example applications. Parasitic power may be a possibility here. These networks will have little control over density and so may require more powerful or adaptive radio transmission to overcome frequent un-

reachability. The rapidity of movement is also highly variable depending on application.

In other networks, node positions may be relatively static. After random deployment by airplane, artillery, or scattering, nodes remain in place unless disturbed by environmental agents. For these networks, mobility is defined in terms of the events that can be sensed by nodes.

Sensor nodes may individually detect the presence of heat, light, motion, magnetism, sound, or other environmental characteristics. Measurements could be sent back to a remote monitoring station via special base stations or uplink nodes in the network.

Smarter sensors could use local coordination algorithms to lessen the amount of raw data transmitted through the network. Aggregation, triangulation, actuation, and other forms of data fusion and analysis define next-generation smart sensor networks.

These cooperative sensor networks could track agents or events as they move through the network, performing network-specific collaboration, analysis, or actuation. Nodes “ahead” of the event could be preemptively awakened from low duty-cycle operation and recruited to join an interested team. The team could follow the event, periodically reporting processed data to a remote monitor.

Remote applications desiring to query or control the moving team face several challenges. The first is addressing the team at the network layer. Since the team is mobile but the nodes are not, no single node may serve as a contact for the team. Also, delivering messages to all nodes in the team is difficult. The membership of the team is constantly changing, and sending updates to the monitor would waste a lot of bandwidth and energy. The updates would be out of date by the time the monitor used them.

This paper describes a suite of modules and their implementations for event tracking applications in sensor networks. A Team/Leader Management (TLM) module provides a team formation and maintenance service. The Tracking Transport Protocol (TTP) provides a basic end-to-end transport service between distributed teams. Both modules use the services of a generic Group Management (GM) module, which can implement sensor-network appropriate group protocols.

The remainder of this paper is organized as follows. We review related work in Section 2. An overview of event tracking and details of the TLM and TTP protocols follows in Section 3. Details of our simulation and results are discussed in Section 4. Finally, we

explore future work and conclude in Sections 5 and 6.

## 2 Related Work

In Mobile IP [2], mobile hosts are always addressed by their home address. When the host is attached to the Internet elsewhere, a home agent forwards datagrams through a tunnel to its current attachment point.

This forwarding to a care-of address is similar to our mechanism for forwarding along a chain of team leaders. Sending messages via one indirection is more efficient than the chaining our mechanism uses. However, in Mobile IP terms, we effectively update the home agent causing it to follow the mobile endpoint. Even though messages may be forwarded more than once, it is likely to be more efficient than when the mobile endpoint is far away from its home. Recently proposed extensions [3] to Mobile IP attempt to solve this by explicitly updating the sender with the current attachment point.

Another mechanism uses DNS to provide the indirection necessary to support mobility [4]. Mobile hosts update their DNS A-records with their new network address. Subsequent connections use this address after looking up the invariant hostname. The mobile host redirects existing connections by sending a SYN plus a migrate option from the new address. An included token identifies which original connection is being migrated.

Sending redirections after migration is similar to our update mechanism, though for individual hosts and not teams. We note that new connections to the mobile host may fail if the host is moving quicker than DNS changes are propagated.

A more recent protocol, M-TCP by Sultan et al. [5], is a TCP extension that supports client-initiated migration of an active connection to another cooperating server. At initial connection time, the client gets a list of cooperating servers and migration certificates. When migrating, the client contacts another server and presents the certificate. The connection is checkpointed on the original server and state transferred to the new one.

In M-TCP, migration relationships between servers must be determined before any connections are established, since the list is sent to the client upon connection. Also, the client initiates the migration according to some application-dependent policy. In a sensor network, *a priori* migration relationships are

untenable due to the scale of the network and the unpredictability of the moving event. Also, the migration policy is better imposed from the mobile team’s side of the connection.

None of the above protocols are targeted for the resource limited sensor network domain. They consider only reliable point-to-point connections. Our approach provides for event-tracking team management and addressing by remote endpoints.

### 3 Event Tracking Overview

Sensor networks are largely defined by their particular applications. The limited resources of nodes make it unlikely that they will possess a wide variety of unused or generic services or protocol layers. We narrow such a large space to a particular type of network, while attempting to remain as general as possible.

We consider a sensor network dedicated to sensing and tracking events as they move. The particular type of sensors or events is unconstrained; we assume only that their range is limited. In particular, we assume that the communication range of an individual node is at least as large as the sense range of events of interest. We do not limit the types of agents, objects, or environmental conditions they are aware of, referring to them collectively as *events*.

Sensor nodes may perform whatever processing is 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 based on detailed queries from a remote monitor. We assume only that when an event is detected, a group of nearby nodes will be interested in some sort of collaboration.

Communication in the sensor network may occur between any nodes without restriction. However, we assume that communication pertinent to the events being tracked will either follow a query/response model or be primarily periodic and uni-directional. Queries and their responses, if relatively infrequent, may tolerate higher packet or processing overheads as are often required by reliable in-order protocols. Periodic streams will certainly be better suited to lightweight, low-cost transport. Their inherent redundancy can be used to advantage for energy savings in the network.

The Team/Leader Management (TLM) module provides a team formation and maintenance service. *Teams* form around events as they are detected in

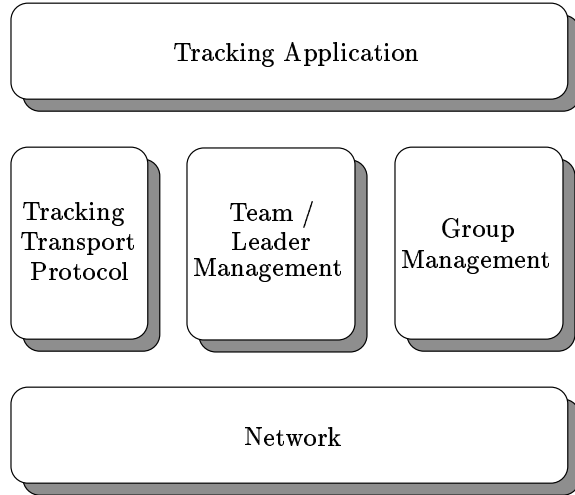


Figure 1: Event tracking modules in a sensor node stack

the network. Nodes join the team as they are recruited, and leave when the event moves away from them. The team may extend beyond the sensory horizon, to awaken nearby nodes and turn on their sensors. Nodes within the sensing region volunteer to be the team *leader*, who is responsible for active recruitment. As the team follows the event, leadership changes and is always among the nodes that can sense the event.

This leadership scheme differentiates among nodes only according to their current role in the team. We do not assume any difference in hardware or software. That the TLM uses a leader also does not impose any requirements for similar methods to be used by the application. Applications can manage their interactions according to any scheme they desire.

We include a generic Group Management (GM) module in our discussion, without specifying the particular sensor network appropriate algorithms or protocols in use. Our goal is to use any such module as is available, the definition of which is outside the scope of this paper.

Our Tracking Transport Protocol (TTP) provides basic end-to-end communication to or between teams. A team first registers its interest or availability by distributing information about its own team and leader throughout the network in an application-specific way. This could be by area multicast, flooding, or other propagation means. Having received a registration, other teams may then send messages to the team via the leader. Leaders act as forwarding routers when a team moves away. Each TTP message

contains the current leader of the team, so future return messages are forwarded as close to the team as possible.

The following sections discuss the details of the APIs and implementations of the TLM, GM, and TTP modules. Figure 1 shows the relationship of these modules at a high-level.

### 3.1 Group Management

The Group Management (GM) module provides group membership and messaging semantics appropriate to a sensor network. These protocols are a subject of ongoing research and are not within the scope of this paper. Other than specifying the interface and basic semantics provided by the GM, we do not constrain its implementation. The TLM and TTP will flexibly work with any reasonable GM.

#### 3.1.1 Interface

- *create()* returns *Group ID*  
A new group is created by calling *create()*. The ID of the group returned should be globally unique, at least probabilistically.
- *join (Group ID)*  
Existing groups may be joined by calling *join()*. Nodes may belong to multiple groups.
- *leave (Group ID)*  
Nodes are removed from membership in a group by calling *leave()*.
- *send (Group ID, Data)*  
A member of a group may send messages to other group members by calling *send()*. The particular semantics of message delivery are unspecified.
- *member (Group ID, Node ID)* returns *boolean*  
Membership of a node in a group can be determined by calling *member()*. It is unspecified whether nodes are able to determine the membership of other nodes. However, a node may always determine whether it is itself a member.

### 3.2 Team/Leader Management

The purpose of the TLM is to form a team of interested nodes around a sensor network event and to maintain the team as the event moves. Nodes beyond sensor range are also recruited for a few reasons.

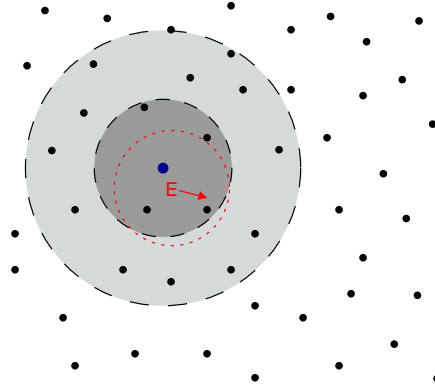


Figure 2: A team forms around a detected event

First, they may need to be awakened or notified of the particular type of event to be sensitive to. They may also be able to contribute in application-dependent analyses. Finally, they provide some smoothing for group formation if the sensors are imperfect or faulty, or if the event moves erratically for other reasons.

Applications may specify how far beyond the sensor range the group is propagated. As discussed before, this should be beyond the sense range, but not so large that the cost of team communication is unacceptable. We specify the team size by hopcount, but other methods based on geographic distance could be used as well.

Figure 2 shows an event, its direction of travel, and a team that has formed to track it. The dotted ring around the event represents the sensor range, inside of which nodes can directly detect the event.

At the center of the shaded region is the current leader. When the event becomes significant or moves into range of the sensor network, the first node to sense it volunteers to be the leader and creates a group via the GM module.

Periodically the leader broadcasts a recruit message, which is propagated for an application-determined number of hops. It contains the event of interest, the group identifier, and the current leader. Figure 2 shows a hopcount of two; the inner shaded region is within one hop, and the outer region is two hops away.

Nodes receiving the recruit message for the first time pass it up to the application, which can decide whether to accept the recruitment. If accepted, the TLM joins the team via the GM module. Subsequent recruit messages keep the team updated with the current leader.

When the current leader can no longer sense the

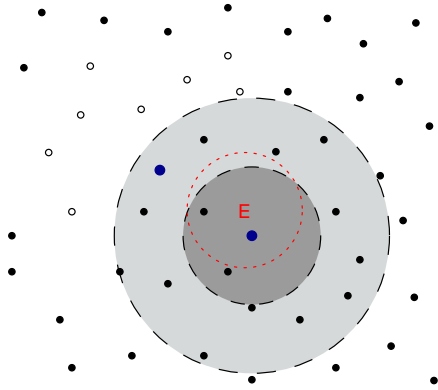


Figure 3: The team and leader follow the event

event, it stops sending the regular recruit messages. When team members fail to receive a recruit message within a certain time, a new leader must be chosen. Leadership is restricted to nodes which can directly sense the event. After a random delay, one such node will volunteer to be leader by sending recruit messages.

Figure 3 shows the team after the event has moved and a new leader has volunteered.

If a node stops hearing recruit messages and cannot directly sense the event, it must wait for an additional time. If no recruit message is received by a new leader, the node must assume the event has moved entirely out of range. It will leave the group via the GM module and return to normal status. These are depicted as hollow circles in Figure 3.

More details about the TLM data structures, application interface, and internal behavior follow. Figure 4 shows a typical interaction sequence between the TLM and other modules.

### 3.2.1 Data Structures

- *EventSeen*:  $\langle \text{Event ID} \rightarrow \text{boolean} \rangle$   
A table in the TLM keeps track of what event IDs are currently able to be sensed by the application. It is updated implicitly by *recruit()* and explicitly by *eventDetected()*.
- *Teams*:  $\langle \text{Team ID} \rightarrow \text{Event ID}, \text{Leader ID}, \text{Hop Count} \rangle$   
The *Teams* table stores leader and event IDs for all known teams. A LRU replacement policy is applied, so leader information for previous team membership is preserved for as long as possible. The size of the group (*Hop Count*) is also stored.

- *RecruitQ*:  $\langle \text{RecruitMsg} \rangle$   
A queue stores incoming recruit messages that reference teams of which this node is not a member. Notifications are also enqueued for the application when the node leaves the team.

### 3.2.2 Interface

- *recruit* (*Event ID*, *Hop Count*, *Data*) returns *Team ID*

Applications indicate their desire to track an event by calling *recruit()*. The TLM marks the event as “seen” in the *EventSeen* table and creates a new team via the GM module. The *Teams* table is populated using the new team ID and this node’s ID as the leader.

A recruit message is broadcasted according to the given *Hop Count* and contains the application’s *Data* as payload. The *SendTimer* is started.

- *receive()* returns *Team ID*, *Data*

Applications dequeue outstanding recruit messages by calling *receive()*. The recruiting team ID and application payload is returned.

The application can then decide whether to join the team by calling *accept()* or by ignoring the message.

Team leave notifications are also delivered to the application through *receive()*, and require no response.

- *accept* (*Team ID*)

Applications register their acceptance of a prior recruit message by calling *accept()*. The TLM joins the team via the GM module, and sets a *ReceiveTimer*.

- *eventDetect* (*Event ID*, *boolean*)

When an event is no longer able to be sensed by the node, the application calls *eventDetect()* with a *false* argument to inform the TLM. This disqualifies the node for leadership, but not membership, in the team.

The TLM sets the flag in the *EventSeen* table accordingly. Note that *recruit()* implicitly sets the flag to *true*.

- *getLeader* (*Team ID*) returns *Leader ID*

The TLM provides an interface for other modules to query the leader of a team. The node

need not belong to the queried team. If the information is still available in the *Teams* table, it is returned.

### 3.2.3 Messages

- *Recruit*:  $\langle Team\ ID, Event\ ID, Leader\ ID \rangle$

When a recruit message is received by the TLM from the network, it checks with the GM module to determine whether this node is already a member of the team. If not, the message is enqueued on the *RecruitQ*, and its data is saved in the *Teams* table.

If the node is already a member of the team, the *Teams* table is updated if the leader has changed. The *ReceiveTimer* is set and any other timers are cancelled.

### 3.2.4 Timers

- *SendTimer*

Expiry of the *SendTimer* means it is time for the leader to broadcast another recruit message, if it still eligible to do so.

If the *EventSeen* table still indicates that this node can sense the event, it will remain the leader. In this case, the TLM sends a recruit message and resets the *SendTimer*.

If this node is now ineligible for leadership, the TLM sets the *LeaveTimer*.

- *ReceiveTimer*

Expiry of the *ReceiveTimer* indicates that this team member has not received a recruit message and that a new leader must volunteer.

The TLM checks the *EventSeen* table to see if this node can sense the event. If not, it sets the *LeaveTimer*, hoping to be recruited before it must leave the team.

If the node is eligible to lead the team, it sets the *VolunteerTimer* to randomly delay before becoming leader. This is to reduce leadership collisions.

Note: The *ReceiveTimer* must be longer than the *SendTimer*.

- *VolunteerTimer*

At expiry of the *VolunteerTimer*, a recruit message is broadcast by the TLM and the *Teams* table is updated to record this node as the leader of the team.

- *LeaveTimer*

When the *LeaveTimer* expires, no recruit message has been received from the team. The TLM leaves the team via the GM module and places a notification message in the *RecruitQ* for the application.

### 3.2.5 Interaction Sequence

Figure 4 details a possible interaction sequence involving the TLM. It is briefly described here to show an example use of the TLM and GM module APIs.

1. The application decides that an event of interest has occurred.
2. Since it is not already in a team, it calls *recruit()*.
3. The TLM calls *create()* to make a new team.
4. It then sends a recruit message describing the team to its neighbors, and sets the send timer.
5. Neighbors within a certain hop-count receive the recruitment.
6. The TLM updates its leader mapping. If the node is not a member already, it puts the recruit message on the queue.
7. The application *receive()*s the recruit message.
8. Deciding that it does want to join the team, it calls *accept()*.
9. The TLM *join()*s the group and sets a receive timer.
10. At some point, the leader moves out of sensor range, as the original leader has in Figure 3. Its application notifies the TLM that the event is no longer being detected by passing a flag to *event-Detected()*.
11. Since the previous leader does not send out recruit messages, some other node notices and volunteers to be the leader, such as the new leader shown in Figure 3. It sends out its own recruit message to its neighbors.
12. Some other node that is now out of range of the team, such as the hollow circles in Figure 3, does not receive a recruit message and cannot itself sense the event. It *leave()*s the team and puts a leave notification in the queue.
13. The application is notified by *receive()* that it is no longer a member of the team.

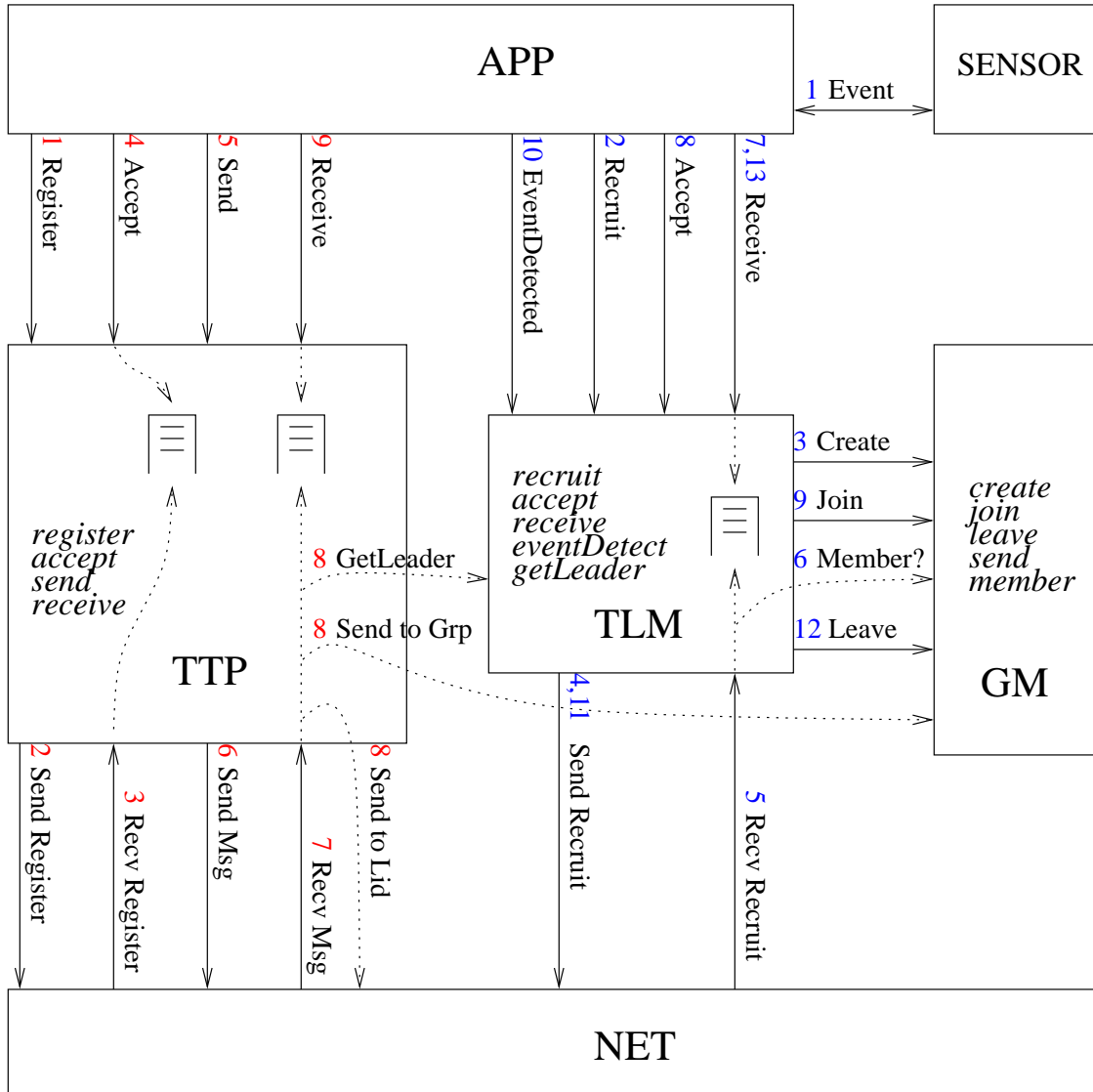


Figure 4: Detailed module interactions

### 3.3 Tracking Transport Protocol

The TTP provides basic end-to-end transport service between distributed endpoints. Lightweight protocols are desirable in sensor networks because of the relatively high cost of communication. Connections in the TTP keep a minimum of state and impose little packet overhead on messages exchanged.

Connections are identified by a  $\langle Team\ ID, Port\ Num \rangle$  pair. Team IDs are assumed to be unique, at least probabilistically, in the network. Port IDs denote only application level semantics. The TTP addresses outgoing messages to the last-known leader of the remote team, which is stored locally in a table.

The required addressing presents an initialization problem: how to know the ID and leader of a remote team before receiving a message from it? To solve this, the TTP allows teams to register themselves with nodes in some area of the network. The precise delivery semantics depend on what the network-layer provides: area-multicast, area-anycast, flooding, gossiping, etc.

Registrations received by the TTP module on the other nodes are enqueued for the application. The team and leader IDs are also stored in a table for future reference.

No handshaking or acknowledgements are required to establish the connection. Having received and

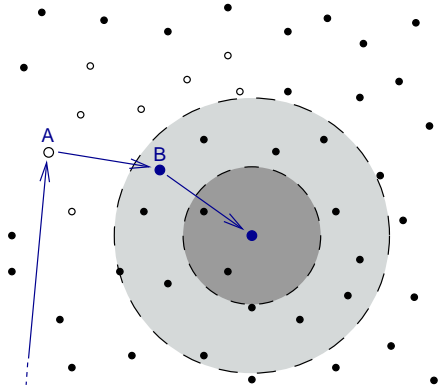


Figure 5: Messages are forwarded to leaders

accepted a registration, the application may subsequently send messages from the local team to the registered remote team. The TTP addresses such a message to the last-known leader of the team.

Registrations also contain application-level data, which may specify the precise types of events the team is interested in. It may also give an expiration time for the registration if appropriate.

Outgoing messages identify the source's current leader in the message header. 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.

The TTP queries the GM module to determine whether this node is a member of the message's destination team. If so, it also queries the TLM to see whether this node is the leader of the team. If this node is the leader, the message is enqueued for the application and propagated to other members of the team by use of the GM module. Non-members and non-leaders forward the message to the leader returned by the TLM query, as depicted in Figure 5.

The semantics provided by the GM module's *send()* primitive determine who in the team may send messages to a remote endpoint. If all members of the team receive messages, at least on a best-effort basis, then all may return responses since their leader information will be updated regularly. If only leaders receive the messages, only they may reply, since other team members will not know to what remote leader to address a message.

Leadership information is retained in the TTP for as long as possible, given limited table sizes. Replacement is done on a least-recently-used basis. This ensures that messages from moderately out-of-date

remote senders can be forwarded along a chain of past leaders until it finally arrives at the current team leader.

Figure 5 shows a remote message addressed to an out-of-date leader, node A. It forwards the message to its own last-known leader, node B, who can then forward it to the current leader. If the current leader replies, the remote endpoint will address subsequent messages directly to it, avoiding the forwarding chain.

By specifying that the team has a radius of at least twice that of the sensor range, previous leaders are guaranteed to be updated with next leader in sequence.

In the following sections, *Source* and *Destination* refer to the  $\langle Team ID, Port Num \rangle$  pair that identifies an endpoint.

### 3.3.1 Data Structures

- *RemoteTeams*:  $\langle Team ID \rightarrow Port ID, Leader ID \rangle$

This table in the TTP maintains the last-known leader information for every remote endpoint from whom it has received a registration or data message.

- *RegisterQ*:  $\langle RegisterMsg \rangle$

Registration messages are enqueued for the application in the *RegisterQ*.

- *MessageQ*:  $\langle DataMsg \rangle$

Regular data messages are kept in a separate queue, the *MessageQ*, for the application.

### 3.3.2 Interface

- *register* (*Source*, *Network Destination*, *Data*)

An application sends a registration by calling *register()*. The local team and port are included in the registration header, and the user-specified data is included in the payload. The leader of the local team is also included, and is obtained from the TLM module's *getLeader()* interface.

The interpretation of the destination depends on what delivery services are provided by the network layer, as described above. *Network Destination* does not refer to a team or leader; it may refer to an area, the whole network, individuals, or some other collection of nodes.

- *accept (Source Mask)* returns *Source, Data*

Incoming registrations are placed on the *RegisterQ*, where they await delivery to the application. The application removes registrations that match the team and port masks specified in the argument to *accept()*.

The source of the registration, team and port, as well as any application-level payload is returned to the calling application.

- *send (Source, Destination, Data)*

Applications communicate with a remote endpoint by calling *send()* and specifying both the local endpoint, the remote endpoint, and the application-level data.

The outgoing data message is addressed at the network layer to the remote leader, obtained from the TLM. The local leader, identically obtained, is also included in the header. Finally, a flags field indicates the message to be of *Remote* origin.

- *receive (Source Mask, Destination Mask)* returns *Data*

Incoming data messages are enqueued on the *MessageQ* for delivery to the application. The application calls *receive()*, specifying masks to match both local and remote endpoints' team and port identifiers.

Matching data messages in the queue are returned to the application.

### 3.3.3 Messages

- *RegisterMsg: <Team ID, Port Num, Leader ID, Data>*

Registration messages are enqueued in the *RegisterQ* after the group to leader and port mapping is saved in the *RemoteTeams* table.

- *DataMsg: <Source, Leader ID, Destination, Flags, Data>*

Data message handling begins with updating the *RemoteTeams* table if the message header indicates a new remote leader.

If the flags field indicates the message is of *Local* origin, it is enqueued in the *MessageQ* and processing stops.

For *Remote* messages, the leader of the destination team is obtained from the TLM's *getLeader()* interface. If this node is not the

leader, the message is sent back to the network layer, addressed for the returned leader.

If this node is the leader of the destination team: it enqueues a copy of the message in the *MessageQ* for the application; it changes the flags field to indicate *Local* origin; and, it sends the copy to the members of the team using the GM module's *send()* interface. Precise delivery semantics are dependent upon the GM.

### 3.3.4 Interaction Sequence

Figure 4 details a possible interaction sequence involving the TTP. It is briefly described here to show an example use of the TTP module API.

1. An application, such as a base station, decides to register its local team (of perhaps only itself) with the rest of the network by calling *register()*.
2. The TTP obtains the local leader (which may be this node) from the TLM and sends the message to some area of the network.
3. Sensor nodes throughout the network receive the registration, save it in their table, and enqueue it.
4. An application on one such node receives the registration by calling *accept()*.
5. The application may now decide to send a message to the remote group that registered, by calling *send()*.
6. The TTP includes the current local leader (using the TLM) and addresses the message to the remote leader from its own table.
7. A node that is a current or former leader receives the message.
8. Local messages are enqueued for the application. If it is not the current leader, it forwards the remote message to its local leader. The local leader enqueues a copy of the message, marks the message local, and sends it to the team via the GM module.
9. The application receives the message.

## 4 Simulation

To fully understand and validate our proposed architecture, we implement our design in a wireless network simulator. Although any simulator is not capable of providing a true analysis of the capabilities of a design, a good simulator serves as a tool to understand and debug proposed problem solutions.

For this work, the simulation environment allows us to apply our modules to the hypothetical problem of event tracking. Specifically, by simulating event tracking in GloMoSim using our mobile transport and team leadership and group formation modules, we are able to analyze the effects of architectural parameters on our model and additionally understand how our modules can be applied to real world problems.

### 4.1 Simulator Details

To test our architecture we use GloMoSim V2.03, a 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.

To handle the overall organization of the simulator, GloMoSim implements a main module responsible for instantiating and organizing nodes, scheduling messages between modules, and tracking the exchange of messages within the simulation. The most important responsibility of the main component is to invoke calls to specific modules as appropriate to control the overall sequential flow of events throughout the simulation. This flow of control is handled by scheduling and passing messages which represent simulation events. These messages pass between abstraction layers (i.e. Radio, Data-Link, MAC, Network, Transport, and Application) to accurately simulate a true sensor network. The issue of how capable GloMoSim is of truly representing a real world environment is beyond the scope of this work.

### 4.2 Scenario

To test our proposed system we implement the event tracking components in GloMoSim. The Transport and Application layers of the simulator are modified

to simulate a hypothetical sensor network environment consisting of nodes communicating over a 15.8 meter radius.

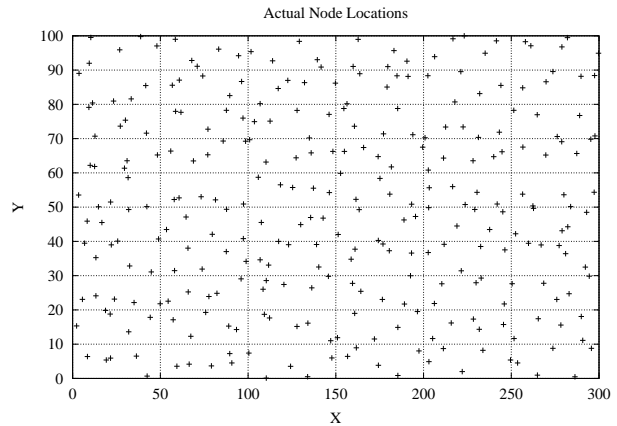


Figure 6: Node locations in 300x100 meter field

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). Three-hundred nodes are uniformly distributed in a 300 x 100 meter field, shown in Figure 6. The simulation time for our experiment is 15 minutes long with an event moving in a straight line across the field at a speed of 0.57 meters/second.

The network is initialized so that every node polls their environment for an event and reports the results every second. The lower level protocols of our wireless network include Geographic Forwarding over IP in the Network layer and 802.11 for the MAC layer.

We simplify the application layer to focus evaluation on our suite of modules. Ordinarily, the TLM's leader need not be related to an application-layer leader, if one even exists. An application interested in determining the position of an event could perform triangulation or other analyses to achieve accuracy, and any number of nodes in the team could report their findings to the base station.

To verify that the team is following the event, we have only the node that is currently the TLM leader report its own location to the base station. We measure the reported locations of the leaders and the actual location of the event. The error calculated from these positions is not intended to show optimality—that depends on using application-layer algorithms for location determination—but the ability of the TLM to follow the event within certain bounds.

To fine tune our simulation we modify several control-

lable and potentially uncontrollable parameter configurations that we would see in a real world application. These include:

- *Sensor Range* of our nodes' sensors (meters)  
The various types and models of sensors available will have different sensor ranges. This is dependent on the mode of propagation and the sensitivity of the sensors.
- *Communication Range* of our nodes' radios (meters)  
We assume a fixed communication range, though networks could also adapt transmission power to compensate for local density variance.
- *Event Speed* through the sensor network (meters / second)
- *Application Polling Period* (seconds)  
This defines how often a node polls its sensor to check for a detected event.
- *Application Report Period* (seconds)  
This defines how often the application reports events to the base station.
- *TLM Hop-count*, the number of hops to propagate team recruit messages
- *TLM Recruit Send Period* (seconds)  
This defines how often the TLM leader sends recruit messages to the team.
- *TLM Recruit Receive Period* (seconds)  
This defines how long a team member will wait before considering taking over team leadership. This must be greater than the *Recruit Send Period*.
- *TLM Leave Wait Period* (seconds)  
This defines how long a team member that is ineligible for leadership should wait before leaving the team after not receiving a recruit message. It must be greater than the *Recruit Receive Period*.

### 4.3 Results

We present results from simulations performed with the following parameter settings:

- *Sensor Range*: 6, 8, 10 meters (variable parameter)

- *Communication Range*: 15.8 meters
- *Event Speed*: 0.57 meters / second
- *Polling Period*: 1 second
- *Report Period*: 1 second
- *Hop-count*: 3 hops
- *Recruit Send Period*: 4 seconds
- *Recruit Receive Period*: 5 seconds
- *Leave Wait Period*: 7 seconds

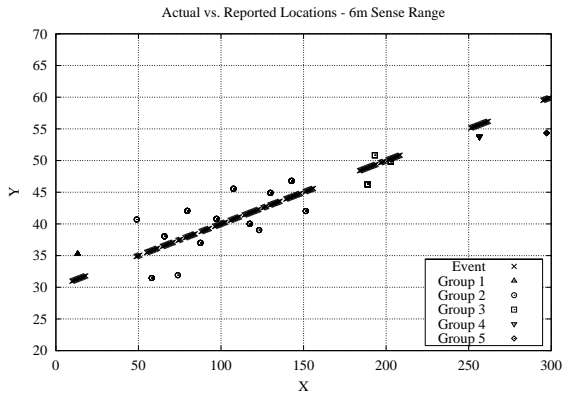
We varied the sensing radius across three different values while keeping the other parameters constant to investigate the effect of the ratio of sensing range to communication range for our architecture. Alternate experiments (not analyzed here) looked at the effect of variations in hop-count, recruit send to recruit receive period ratios, event speed, and report period. We find that these parameters affect simulation results in similar ways, and we present results that show the general conclusions.

Figures 7(a), 7(c), and 7(e) illustrate the relationship between actual event location and the reported (leader) location. The figures plot the actual event location against its reported location (as seen by the base station) for a sensing range of 6, 8, and 10 meters respectively.

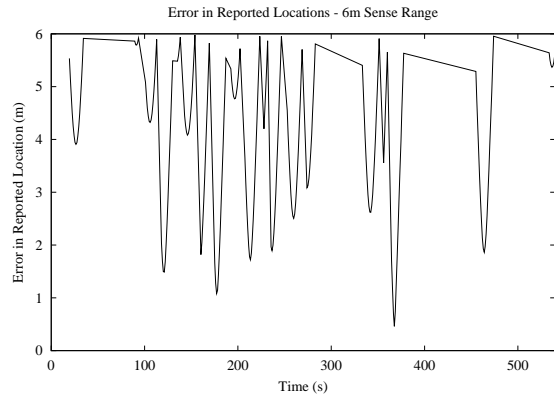
As can be seen in Figure 7(a), event tracking with a sensor range of 6 meters produces significant "holes" in the data during tracking. When sensor range is small relative to node density (as shown in this scenario), situations occur where the event is not sensed by any node. Even though the surrounding nodes exist in a team around the event, none, including the leader, can actively sense the event and therefore will not assume leadership. As a result, the team collapses after the leave wait period for all team members. Later, when the event is sensed by some other node along the path, team formation is initiated by that node and tracking continues within a new team. This is illustrated in Figure 7(a), where 5 separate teams form and track the event.

Only one "hole" is shown in Figure 7(c) for the 8 meter sensor range, indicating that the network density is just below what is necessary for team continuity.

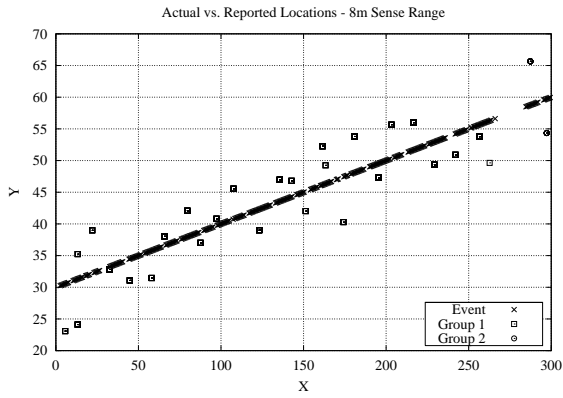
When the sensor range is extended to 10 meters, this problem disappears. In these cases, the "gaps" between member nodes are very narrow or altogether eliminated. Team continuity is not lost because a new leader emerges when a previous leader stops sending



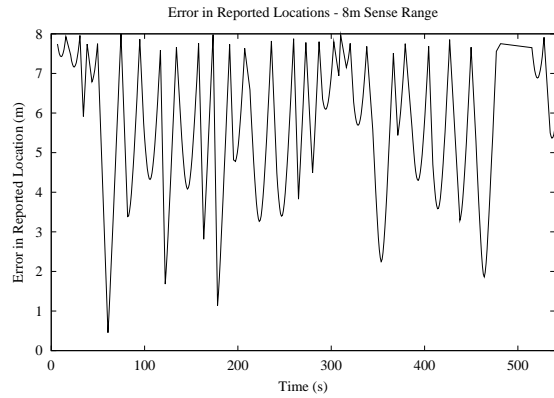
(a) Reported positions for 6m sense range



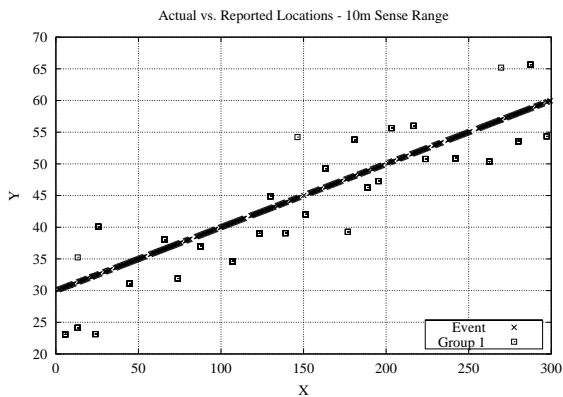
(b) Report error for 6m sense range



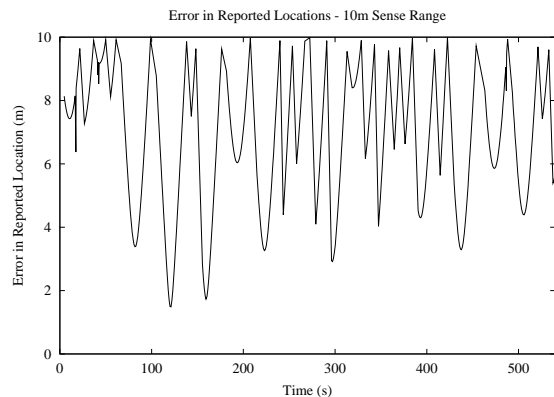
(c) Reported positions for 8m sense range



(d) Report error for 8m sense range



(e) Reported positions for 10m sense range



(f) Report error for 10m sense range

Figure 7: Simulation results

recruit messages. Figure 7(e) illustrates that a single team tracks the event throughout. In this case, the small breaks in reporting happen only during the period when leadership is transitioning.

Figures 7(b), 7(d), and 7(f) illustrate the error in reported value for sense ranges of 6, 8 and 10 meters respectively during the course of the event's movement through the sensor network. Since the reported value of the event's location is the leader's location, the error in reported value is a function of the distance of the event from the leader.

When a node becomes the leader of a team, the event will be some distance within the sensor range away from it. As the event moves closer to the node, the leader's reported location will better approximate the event's location and hence error decreases, though this will only occur if the new leader is in front of the event. If a leader volunteers "behind" the event or after the event passes the leader and starts moving away from it, error in reported location increases until the leader loses it's sense of the event. At this point the last reported error will be close to the sensor range. Viewed on a graph, this appears as a parabola when error is plotted against time. Spikes in this graph show leadership changes within a team.

Figure 7(b) illustrates the error for a sense range of 6 meters. The parabola described above occurs for each leader. Hence, leadership changes occur at the peaks of the graph.

Irregularly sloped lines represent a time when leadership transition has failed and no node is reporting the event. As can be seen, the peaks have maximum values that approach the sensor range. No values above the sensor range can be reported since a node not sensing an event will not report this event to the base station.

Figures 7(d) and 7(f) similarly illustrate the errors for sensor ranges of 8 and 10 meters, respectively. Since there are no "holes" in these results, the graph oscillates between peaks throughout the course of the event's movement. The majority of the peaks occur close to 8 or 10 meters which are the respective sensor ranges.

## 5 Future Work

The Tracking Transport Protocol addresses the lightweight, best-effort messaging that is likely to dominate sensor network communication. Some applications may require more guarantees than the TTP

provides. Future work could add a companion protocol to provide reliable and sequenced messaging to address this need.

Our simulation results show promise for the usefulness of our protocols. Validation on an actual platform, such as Berkeley's motes [1], may expose unexpected challenges or opportunities for further refinement.

Real sensor networks may be able to detect different types of sensor events, and may have multiple instances of any one particular event type. Such networks may see events cross paths, merge, or divide. In the future we could more completely characterize the behavior of our team formation and maintenance mechanism in a more elaborate scenario possessing these complicating activities.

## 6 Conclusion

Large-scale sensor networks require unique approaches to distributed team formation and control. Their resources are too limited to support traditional techniques applicable to wired or high-power wireless systems. Lightweight protocols and algorithms are needed to minimize computation and especially communication.

We have presented a suite of modules for suite of modules for distributed team formation, maintenance, and communication, oriented around the goal of tracking mobile events.

The Team/Leader Management module handles the formation and ongoing maintenance of a team and its leader, ensuring that leadership changes as necessary to stay within sensor range of an event. It uses the services of a generic Group Management module, which may use sensor-network appropriate protocols. The Tracking Transport Protocol allows distributed endpoints to register with nodes in the network and to subsequently communicate despite team membership changes and mobility. It uses a self-updating forwarding mechanism to find the current location of the team.

Simulation results show that through a well defined API, our modules are capable of supporting a simple event tracking application with error defined by the sensor range. Specifically, we have shown that with reasonable assumptions, the TLM is able to form teams around events and perform leadership maintenance in a way that enables the TTP to provide communication between mobile distributed endpoints.

Our solution separates the concerns of end to end communication from team formation and leadership management. Since group semantics are provided by the generic GM module, they can be replaced and optimized within the constraints of its API. Teams may also form for entirely local purposes, in which case the TTP is not needed.

Though the TLM tracks events to provide its service, it can support any type of application that is interested in collaborating with a team of its peers about such events, even if they have no need for explicit application-level event tracking.

## References

- [1] Jason Hill, Robert Szewczyk, Alec Wood, Seth Hollar, David Culler, and Kristofer Pister, "System Architecture Directions for Networked Sensors," In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, Cambridge, M.A., November 2000.
- [2] C. Perkins, Ed., "IP Mobility Support," RFC 2002, *IETF*, Oct. 1996.
- [3] C. Perkins and D. Johnson, "Route Optimization in Mobile IP," Internet Draft, *IETF*, Sep. 2001.
- [4] A.C. Snoeren and H. Balakrishnan, "An end-to-end approach to host mobility," In *Proc. of MobiCom 2000*, August 2000.
- [5] Florin Sultan, Kiran Srinivasan, Deepa Iyer, Liviu Iftode, "Migratory TCP: Connection Migration for Service Continuity in the Internet," To appear in *ICDCS 2002*.