Design and Comparison of Lightweight Group Management Strategies in EnviroSuite[⋆]

Liqian Luo, Tarek Abdelzaher, Tian He, and John A. Stankovic

Department of Computer Science, University of Virginia,
Charlottesville, VA 22904
{ll4p, zaher, th7c, stankovic}@cs.virginia.edu

Abstract. Tracking is one of the major applications of wireless sensor networks. *EnviroSuite*, as a programming paradigm, provides a comprehensive solution for programming tracking applications, wherein moving environmental targets are uniquely and identically mapped to logical objects to raise the level of programming abstraction. Such mapping is done through distributed group management algorithms, which organize nodes in the vicinity of targets into groups, and maintain the uniqueness and identity of target representation such that each target is given a consistent name. Challenged by tracking fast-moving targets, this paper explores, in a systematic way, various group management optimizations including semi-dynamic leader election, piggy-backed heartbeats, and implicit leader election. The resulting tracking protocol, *Lightweight EnviroSuite*, is integrated into a surveillance system. Empirical performance evaluation on a network of 200 XSM motes shows that, due to these optimizations, Lightweight EnviroSuite is able to track targets more than 3 times faster than the fastest targets trackable by the original EnviroSuite even when 20% of nodes fail.

1 Introduction

The increasing popularity of sensor networks in large-scale applications such as environmental monitoring and military surveillance motivates new high-level abstractions for programming-in-the-large. As a result, several programming models that encode the overall network behavior (rather than per-node behaviors) have been proposed in recent years. Examples include virtual machines [1][2], and database-centric [3][4][5], space-centric [6][7], group-based [8][9], and environment-based [10] programming models, which offer virtual machine instruction sets, queries, sensor node groups and environmental events, respectively, as the underlying abstractions with which the programmer operates. These abstractions capture the unique properties of distributed wireless sensor networks and expedite software development.

[⋆] The work reported in this paper is funded in part by NSF under grants CCR-0208769 and ITR EIA-0205327.

An important category of sensor network applications involves tracking environmental targets. In these applications, some internal representation of the external tracked entity is maintained such as a state record, a logical agent, or a logical object representing the physical target. A given target in the environment should be represented uniquely and its identity should be preserved consistently over time. One way to ensure unique, consistent representation is to relay all sensor readings to a centralized base-station which runs spatial and temporal correlation algorithms to infer the presence of targets, assign them unique identities, and maintain such identities consistently. Such a centralized approach, however, is both inefficient and vulnerable. In addition to relying on a single point of failure, it results in excessive power consumption due to communication with a centralized bottleneck and may unduly increase latency, especially when targets move far away from the base-station.

To avoid these limitations, in EnviroSuite, we take the alternative approach of processing target data at or near the location where the target is sensed. Hence, appropriate distributed group management policies are needed to ensure the uniqueness and identity of target representation such that targets are given consistent names and sensors agree on which target they are sensing. This paper systematically investigates different system optimizations in the design of such group management algorithms in a sensor network. The resulting tracking system, *Lightweight EnviroSuite*, is used in a sensor network surveillance prototype that has since been transferred to the Defense Intelligence Agency (DIA). It is evaluated on a sensor network of 200 XSM motes [11]. Results from field tests of the overall system are provided, focusing on tracking performance. (Other performance aspects of the system such as efficacy of energy management algorithms will be reported elsewhere.) It is seen that realistic targets can indeed be tracked correctly despite environmental noise using low-range sensors. Our field test results show that even when 20% of nodes fail, Lightweight EnviroSuite is able to track targets more than 3 times faster than the fastest targets trackable by the original EnviroSuite, due to the optimizations described in this paper. The improved tracking coincides with reduced communication cost.

The remainder of the paper is organized as follows. Section 2 presents the background information and enumerates limitations of the original EnviroSuite. Section 3 explores group management strategies and their effects that constitute Lightweight EnviroSuite. Section 4 analyzes the performance results of a surveillance system that is constructed from Lightweight EnviroSuite. Finally, Section 5 supplies a summary and concludes the paper.

2 Background

Target tracking has received special attention in recent ad hoc and sensor networks literature. Many prior approaches (e.g., in the ubiquitous computing and communication domains) focused on tracking cooperative targets. Cooperative targets are those that allow themselves to be tracked typically by exporting a unique identifier to the infrastructure (such as a cell-phone number). Examples of cooperative targets include cell phones, RFID tags [12] and smart badges [13]. Since such devices are preconfigured with a unique identity, the tracking problem is generally reduced to that of locating the uniquely identified device and performing hand-offs if needed (e.g., in cellular phones). In contrast, our goal is to track non-cooperative targets such as enemy vehicles that do not broadcast self-identifying information. The presence and identity of such targets can only be inferred from sensory signatures, as opposed to direct communication with the target. Tracking non-cooperative targets is more challenging due to the difficulty in associating sensory signatures with the corresponding targets (e.g., all tanks look the same to our unsophisticated motes). The presence of target mobility further complicates the tracking problem.

One of the first research efforts on group management for non-cooperative target tracking has been conducted by researchers at PARC [14]. Their group management method dynamically organizes sensors into collaborative groups, each of which tracks a single target. Typical tracking problems such as multi-target tracking and tracking crossing targets are solved elegantly in a distributed way. Other approaches to target tracking include [15] which presents a particle filtering style algorithm for tracking using a network of binary sensors which only detect whether the object is moving towards or away from the sensor. A scalable distributed algorithm for computing and maintaining multi-target identity information in described in [16]. In [17] a tree-based approach is proposed to facilitate sensor node collaboration in tracking a mobile target.

The authors have investigated the tracking problem in several of their own prior publications. Similar to [8], in [18] we present a set of group management algorithms which form sensor groups at the locations of environmental events of interest and attach logical identities to the groups. Based on [18], EnviroTrack [19] proposes an environmental computing paradigm which facilitates tracking application development. EnviroSuite [20] further extends the paradigm to support a broader set of applications that are not limited to target tracking. Geared for tracking of fast-moving targets with low communication cost on available hardware platforms that have limited sensing and communication abilities, this paper proposes lightweight group management algorithms for EnviroSuite. Optimizations described in this paper may be applicable to other systems such as [8] as well.

The design of EnviroSuite assumes that each node can independently detect the potential presence of a target (subject to false alarms). For example, the presence of a magnetic signature, motion, and engine sound can be independently detected by each XSM mote to signify the potential presence of a nearby moving vehicle in a desert surveillance scenario. It is further assumed that sensor readings do not interfere with each other. Hence, tracking reduces to the problem of correct mapping of nodes that detect target signatures to the actual physical target identities responsible for these signatures. EnviroSuite therefore organizes nodes that detect targets of interest into groups, each representing one target.

Different from traditional centralized tracking schemes, the data association between targets and groups in EnviroSuite is done in a distributed way. Namely, all nodes that sense a target and can communicate directly assume that they sense the same target and consequently join the same group. The resulting aggregate behavior is that connected regions of sensors that sense the same signature are fused into the same group. Observe that when the target moves, group membership changes reflecting the changing set of sensors that can sense it at the time. A leader is elected for the group among the current members. A leadership hand-off algorithm ensures that group state is passed to each new leader.

In this paper, we focus on point targets (i.e., those approximated by a point in space, such as a vehicle), as opposed diffuse region targets (given by an area, such as a chemical spill). It is assumed that the communication range of a node is sufficiently larger than its sensing range. Hence, all group members sensing a point target are within each other's radio range. Consequently, the data dissemination scheme within each group can simply use local broadcast to share sensory information. The leader performs data fusion in an application-specific manner to collect higher-level target information. Geographic forwarding [21] is used for communication with destinations external to the group. For example, in the evaluation section, each group leader estimates target position by averaging locations of group members and sends the result to remote base-stations periodically. Extensions of this scheme to diffuse region targets are described elsewhere [20].

This section briefly reviews the EnviroSuite programming paradigm and its core component, *MGMP* (*multi-target group management protocol*). It also analyzes and evaluates the limitations of MGMP when facing the practical requirements of a typical surveillance system.

2.1 EnviroSuite Abstractions and Challenges

EnviroSuite [20] is an object-based framework that supports *environmentally immersive programming* for sensor networks. Environmentally immersive programming refers to an object-based paradigm in which logical objects and objects representing physical environmental entities are seamlessly combined. Hence, EnviroSuite differs from other object-based systems in that its objects may be representations of elements in the external environment. At the implementation level, such objects are maintained by the corresponding group leaders. Upon detection of external elements of interest, nodes detecting the element self-organize into a group. The group leader in EnviroSuite dynamically creates an object instance to represent the tracked target. The group management protocol maintains a unique and identical mapping between object instances (or group leaders) and the corresponding environmental elements they track, such that object instances float across the network geographically following the elements they represent. This co-location is ideal for the execution of location sensitive object code that carries out sensing and actuation tasks. Objects encapsulate the aggregate state of the elements they represent (collected and stored by group leaders), making such state available to their methods. They are therefore the units that encapsulate program data, computation, communication, sensing and actuation. Object instances are destroyed when their corresponding environmental elements leave the network. This occurs naturally when the membership of the corresponding sensor group is reduced to zero.

Objects can be point objects (created for mobile targets that dynamically change their geographical locations), region objects (mapped to static or slowly moving regions), or function objects (not mapped to an environmental element). EnviroSuite is able to support both point objects and region objects in the same framework due to their similarities. Namely, (i) the corresponding external elements are detected by a group of geographically continuous nodes, and (ii) the aggregate state of the elements are collected by group leaders. However, the focus of this paper is on target tracking applications. Hence, we discuss mainly maintenance of point objects.

The biggest problem faced in EnviroSuite is the challenge of maintaining a unique and identical mapping between each object and the corresponding environmental element despite of distribution and possible mobility in the environment. In the rest of this paper, a *target* refers to a geographically continuous activity in the physical environment that persists over some interval of time. *Object uniqueness* dictates that each target be represented and that it be represented by exactly one logical object instance. *Object identity* dictates that the mapping between targets and objects be immutable. In other words, a target is always mapped to the same object instance identified by its *object ID*.

The problem of object uniqueness and identity is complicated by several factors. One is the need for seamless object migration across nodes as the target moves. Another is that sensor nodes that become aware of an external target should be able to tell whether it is a target previously seen by other neighboring sensors or not. Otherwise, an incorrect target list will be collectively maintained or an incorrect mapping will result between targets and objects. In the following we describe a solution to these problems.

2.2 MGMP and Its Limitations

The core component of EnviroSuite previously proposed by us is a set of multi-target group management protocols, named MGMP, which resolves the object uniqueness and identity problem. When predefined target signatures are detected by a set of nearby nodes, MGMP reacts by creating a *group* attached with a unique object ID. The set of nodes become group *members*, whose task is to periodically sense, calculate and report predefined object *attributes* (such as temperature, location, etc.) to the group *leader*. These reports are called *member reports*. A single leader is elected among these members to uniquely represent the group as one object to the external world. To avoid electing a node that is imminently going out of sensing range which results in yet another election, it is preferred to elect nodes near the target. Though current leader election doesn't enforce such preference, it can be easily adapted to do so if distances to the target can be inferred from detection results.

The leader is responsible for the maintenance of object attributes. It records member reports keeping only the most recent one from each member. It periodically creates a digest of the reports, and either keeps it as the internal state or

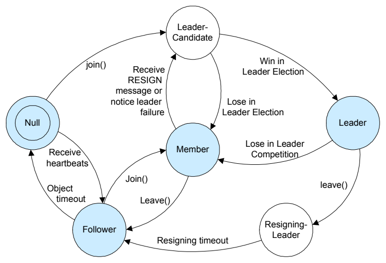

Fig. 1. Node state transitions in MGMP

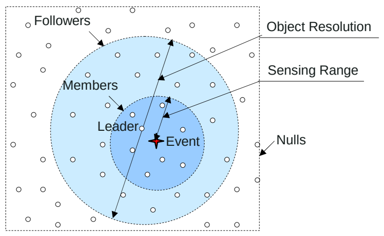

Fig. 2. States of nodes around a target

sends it to the external world (e.g., base stations) based on user specification. These reports are called *leader reports*. The leader is also responsible for object uniqueness and identity maintenance. It periodically sends leader heartbeats to nodes within half an *object resolution* (in meters, defined by users) to advertise the object ID as well as its internal states. By design, half the object resolution must be larger than twice the sensing range such that every member can receive leader heartbeats. Nodes within half the object resolution, which cannot sense, but are aware of the target through heartbeats, are called group *followers* as distinguished from members. Follower nodes are prevented from spawning new groups. Follower nodes are centered around the leader since leader locations provide a good approximation of target positions. Though centering around the target would be a better solution, given the fact that both communication range and sensing range are irregular, the extra complexity required to do so is not worth it.

Nodes dynamically join or leave the group whenever they detect or lose the target. If a leader loses the target, it sends out a `RESIGN` message to request a leadership handoff. Upon the reception of such messages, the most current members reelect a new leader to take over leadership. Figure 1 illustrates the complete node state transitions in MGMP and Figure 2 depicts a typical node state distribution around a target. MGMP employs two important strategies to enhance robustness in the face of failures:

1. **Dynamic Leader Election:** Leaders are always dynamically elected among all current members. There are no pre-designated leader candidates. When leader election starts, each member sets a timer at random from 0 up to *maximum back-off time* and, when the timer expires, claims its leadership by messages, the reception of which terminates other members' timers as well as their unsent messages. This strategy ensures robustness to node failures.
2. **Periodic Leader Heartbeats:** Leaders periodically send out heartbeats so that leader failure can be detected by neighboring nodes.

The main goal of our deployed system is to alert a military command and control unit of the occurrence of targets of interest in hostile regions. Targets of interest may include civilian persons (unarmed), armed persons or vehicles. The system is required to obtain and report current positions of such targets to a remote base station to create tracks in real time. Several application requirements must be satisfied to make this system useful in practice. First, the application must have the ability to track typical military vehicles with velocities varying from 5 mph to 35 mph. Object uniqueness and identity must be ensured. Second, in our application, real-time updates on target trajectory must be sent to a base-station to be used by other devices such as cameras, which requires high accuracy and low reporting latency. Given the severely constrained bandwidth of current mote platforms, the communication cost should be minimized to reduce communication latency and maximize information throughput.

Does MGMP satisfy these requirements? We first try to answer the question through experimental results on TOSSIM [22]; a simulator for TinyOS [23] that emulates the execution of application code on the motes. Our experiments consist of 120 nodes deployed in a 30×4 grid 10 meters apart. Sensing range is set to be 1 grid length and communication range is 3 grid lengths. These settings reflect our real system where sensor devices are deployed in a grid 10 meters apart, sensing range is around 10 meters, and communication range is approximately 30 meters. We simulate a target moving across the field in a straight line to test tracking performance. The target is tracked with a sensor polling period of 0.02 s. Consistent with the real system requirements, members report to leaders their own locations twice a second. Leaders triangulate received locations to estimate target position and report estimations to the base station (located in a corner node) twice a second. The same testbed is used in later sections to evaluate new schemes.

Figure 3 shows the number of objects formed for the single target during its presence in the field. The uniqueness of target representation requires that only one object be formed. As is seen in figure, this is not always the case. The number of objects formed is one at lower target velocities, but it increases as target velocity increases. This violation is due to the difficulty in reaching agreement on target identity quickly enough which leads some sensors to believe that they are seeing different targets. The effects of maximum back-off time are more subtle. As is seen from Figure 3, if maximum back-off time is too small such as 0.2 s, the number of objects generated can be large since multiple nodes may become leaders and create new groups at the same time to represent the same target. If it is too large, fast targets may move out of the sensing range of a node before its back-off timer expires, so that the current object is lost and spurious ones are created. In theory, it is possible to derive the appropriate back-off time analytically for a particular target velocity. The main idea is that leader migration (via election and hand-off) should be faster than target speed for the target to never escape its tracking group. Nevertheless, such a derivation would have to be experimentally validated since it is difficult to account for various imperfections such as the irregularity of the sensing and communication ranges and the non-uniform distributions of nodes in practice.

Observe that no back-off timer value in Figure 3 can maintain object uniqueness at target speeds more than 1 grid length per second (grid/s) or 22 mph (since grid length is 10 meters). These results are far from the desired perfor-

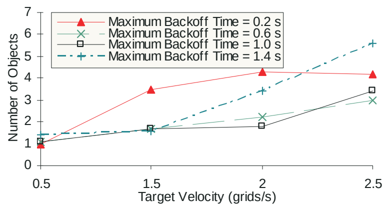

Fig. 3. Number of objects for varied maximum back-off time and target velocities

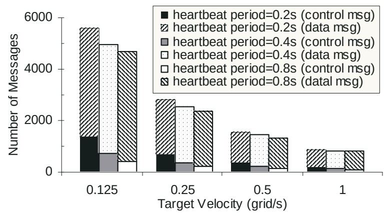

Fig. 4. Number of messages for varied heartbeat periods and target velocities

mance (35 mph). The overhead of dynamic leader election is the main reason why better results cannot be achieved, since long leader-election delays slow down the migration of groups, thus making fast-moving targets untrackable.

Figure 4 depicts the number of *control messages* (leader heartbeats and other group maintenance messages) and *data messages* (member reports and leader reports) sent during the presence of a target. The number of data messages decreases with increasing target velocity, because it is proportional to the duration of target presence due to periodicity of member reports and leader reports. The number of control messages exhibits similar trends since heartbeats that dominate control messages are also periodic. Obviously, control messages also decrease with longer heartbeat periods. However, minimizing communication cost by indefinitely increasing heartbeat periods is not feasible since longer heartbeat periods increase the vulnerability to message loss.

The above observations give insights into improvements to EnviroSuite that enhance tracking performance and reduce communication cost while maintaining robustness to failures. These improvements are described next.

3 Group Management Strategies in Lightweight EnviroSuite

This section explores in more detail the performance problems of current strategies, proposes a series of new strategies, and applies them one by one to EnviroSuite to verify their individual effects on the current system. These new group management strategies, as a whole, constitute a very practical and efficient version of EnviroSuite, called *Lightweight EnviroSuite*.

3.1 Semi-dynamic Leader Election

Dynamic leader election, as the main factor that limits tracking performance of MGMP, affects the maintenance of object uniqueness and identity in two ways. First, it causes long leader handoff delays. In dynamic leader election, all members are competitors for leadership. Hence, consensus has to be achieved among all on a single leader. Obviously, the more members participate, the slower

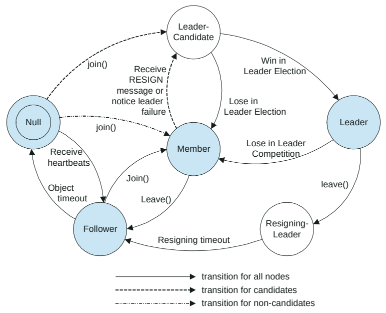

Fig. 5. Node state transitions for semi-dynamic leader election

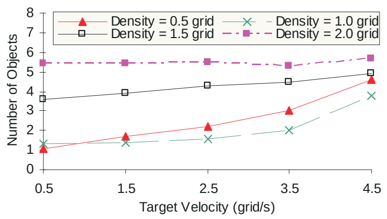

Fig. 6. Number of objects for varied target velocities and candidate densities

the consensus. Second, it increases the possibility of message collisions since all members are exchanging messages to compete for leadership.

A better solution is to allow only a portion of all members to compete for leadership, which we call *semi-dynamic leader election*. Semi-dynamic leader election includes an initialization phase which pre-elects a portion of the nodes to be *candidates* (the potential competitors for leadership); others become *non-candidates*. The pre-election of candidates is similar to dynamic leader election in EnviroSuite. Each node sets a random timer and, when the timer expires, claims itself as a candidate. Nodes within distance x that receive this claim message become non-candidates. Ideally, the algorithm elects at least one candidate within any circular area of radius x. We call this x the *candidate density*. The node state transition changes accordingly as shown in Figure 5. Transitions to `Leader-Candidate` occur only when the corresponding nodes are candidates. `Null` nodes become `Members` instead of `Leader-Candidates` when they are non-candidates. Since only `Leader-Candidate` nodes attend leader election, these changes make candidates the only ones competing for leaderships.

Figure 6 illustrates the number of objects created for targets with different velocities when different candidate densities are set. Semi-dynamic leader election allows for a smaller maximum back-off time (set to 0.2 s in the following experiment) in leader election due to a reduced number of competitors. The $Density = 0.5\,grid$ curve performs the same as dynamic leader election since all nodes are candidates (maximum back-off time is set to 0.6 s for better performance in this case). As seen from Figure 6, a proper candidate density, say 1.0 grid, makes the semi-dynamic scheme outperform the dynamic one.

Observe that, a very low candidate density results in worse performance than dynamic leader election since candidates are so scarce that, in most groups, no leader is elected to maintain objects. We call the phenomenon a *leader desert*. The dark grey circle in Fig. 7 shows a leader desert where no candidate exists. If the target moves further to the right and gets detected by the nearest candidate outside the follower set, a spurious object is created by the candidate since it is not aware of the existing object. Even when candidate density is 1.0 grid, a leader desert still appears occasionally, which hurts object uniqueness slightly.

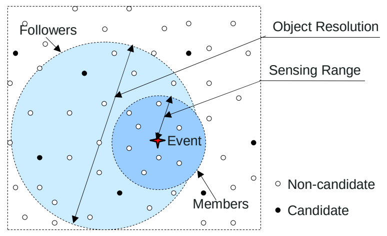

Fig. 7. Leader desert

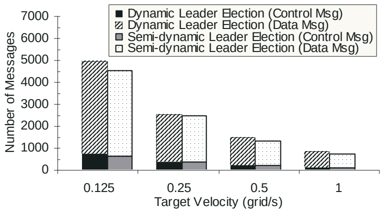

Fig. 8. Comparison of number of messages for different target velocities

This explains why semi-dynamic leader election performs a little worse when target velocity is 0.5 grid/s. The leader desert problem is solved in later sections. As a side-effect, semi-dynamic leader election also results in lower communication cost since fewer control messages are sent to compete for leadership, which is shown in Fig. 8.

On the disadvantage side, robustness to node failures is expected to degrade when using semi-dynamic leader election due to a higher vulnerability to failures of leader candidates. However, this can be partially compensated by executing candidate pre-election more frequently. We discuss the overall failure robustness of the new scheme in later sections.

3.2 Piggy-Backed Heartbeat

Periodic leader heartbeat entails big overheads that are not affordable in applications with severe bandwidth constraints. Yet, it plays the most critical role in MGMP. First, it recruits followers to prevent these boundary nodes from creating spurious groups. Second, its periodicity makes leader failures perceivable, and thus recoverable. Third, the periodicity also improves robustness to message loss. Therefore, the challenge is how to reduce overhead while retaining the advantages of frequent heartbeats.

Fortunately, another component in MGMP exhibits the behavior of sending periodic messages; namely, *object attribute collection*. Members periodically send sensed attribute data to leaders and leaders periodically aggregate received data, process it, and send results to the external world if required. If heartbeats can be piggy-backed into these member reports and leader reports, periodic heartbeat becomes almost free. We call this new scheme *piggy-backed heartbeat*, where heartbeats are transformed into *leader heartbeats* (heartbeats piggy-backed into leader reports) and *member heartbeats* (heartbeats piggy-backed into member reports). Leader desert is no longer an obstacle to object state dissemination, since members take over this task during leader absences. Since object uniqueness is ensured through leader uniqueness, members are only allowed to repeat heartbeats originated from leaders and leaders are still the only authority that may update object information.

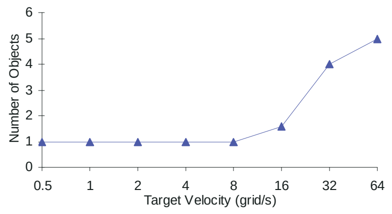

Fig. 9. Object maintenance in semi-dynamic leader election

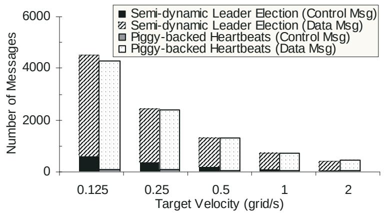

Fig. 10. Comparison of communication overhead

In the piggy-backed heartbeat scheme, member heartbeats and leader heartbeats are treated differently: only the reception of leader heartbeats transits pre-elected candidates from state `Null` to `Follower`, while member heartbeats transits them into an intermediate state, called `Null-Follower`. If a node detects a target while in this state, it transits to Leader-Candidate to compete for leadership. This is unlike a regular `Follower`, which becomes a `Member` upon target detection. Without these changes, member heartbeats may transit all potential leaders to `Follower` state and then to `Member` upon the detection of the target, making the group follow a target without any leaders.

The aforementioned efforts make the piggy-backed heartbeat scheme a big improvement in object maintenance. Compared with the maximum trackable velocity seen in MGMP (1 grid/s), this improved version maintains object uniqueness and identity for targets with velocities up to 8 grid/s as shown in Figure 9. Figure 10 suggests another big improvement in reducing control messages.

3.3 Implicit Leader Election

It is possible to further reduce the protocol costs by employing an *implicit leader election* scheme. An assumption is made in this scheme that monitoring tasks are periodic and that after each period monitoring results are communicated by each tracking group to the external world. This assumption is reasonable since periodicity is a typical property of sensor network applications. The scheme allows candidates to start the execution of leader tasks such as data aggregation, whenever they detect the target. Note that, their task periods are unsynchronized since nodes usually do not begin to sense the target at exactly the same time. As a result, multiple but limited potential leaders are executing tasks in a group. At any point in time, if the node that first reaches the end of a task period sends out a result report, other neighboring nodes including other potential leaders simply accept the results and become *inactive* in their current task periods, which prevents them from reporting the same redundant results when finishing their periods. Hence, the external world sees the illusion of a single group leader.

Figure 11 illustrates an example. Candidate A senses the target from time 0 to 2.5 and B from 0.25 to 3.5. The length of task period is 1. Both A and B are initially active. At time 1, A reaches the end of its period and sends a result report. Receiving this report, B admits A as the current leader, accepts

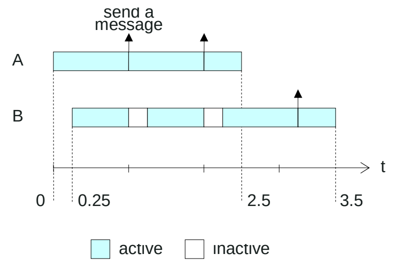

Fig. 11. An example of implicit leader election

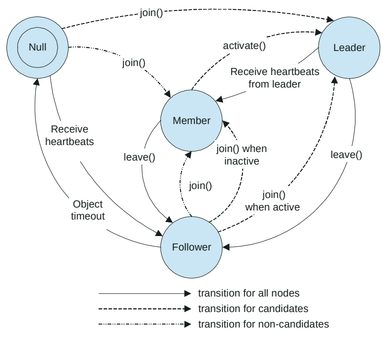

Fig. 12. Node state transitions for implicit leader election

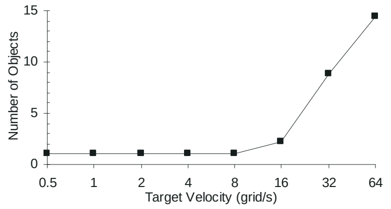

Fig. 13. Object maintenance in implicit leader election

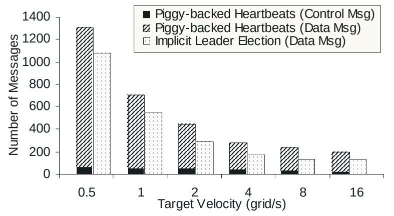

Fig. 14. Comparison of communication overhead

A's results and makes itself inactive, which makes B silent at time 1.25 when it finishes its task period. Similarly B is still silent in the next period since A sends a message first. Finally, A quits the leader competition at time 2.5 when it loses the target. B continues A's work and reports the results at the end of its third period (time 3.25). This way, tasks are executed continuously between different leaders, the results of which are exposed to the external world at a rate ($3/3.5 \approx 0.9$ report/s) that is very near to the defined rate (1 report/s).

Figure 12 depicts the new node state transition graph. `activate()` is called by each node when beginning a new task period. Different from all previous versions, intermediate states (`LeaderCandidate`, `ResigningLeader`) no longer exist since implicit leader election eliminates the need for nodes to stay at the `LeaderCandidate` state sending `CANDIDATE` messages to compete for leadership and to stay at `ResigningLeader` sending `RESIGN` messages to start new leader election. The elimination of control messages further improves communication performance as shown in Figure 14. Meanwhile, a comparable performance in object maintenance (maximum trackable velocity 8 grid/s) is achieved as Figure 13 shows.

Note that implicit leader election can not be applied to applications where duplicate execution of tasks is not allowed or where tasks are not periodic. However, since the EnviroSuite compiler [20] has the ability to dynamically select modules during compilation based on programmer's application definition, a version without implicit leader election can be composed in such cases.

Overall, due to the optimizations mentioned above, Lightweight EnviroSuite achieves roughly an order of magnitude improvement in maximum trackable velocity (8 grid/s compared with 1 grid/s in EnviroSuite). The number of messages per second for targets at 0.5, 1 and 2 grid/s is reduced 25%, 12% and 37%, respectively.

3.4 Failure Tolerance

This section discusses the overall robustness of Lightweight EnviroSuite to typical failures in the sensor network: message loss and node failures. Message loss harms object maintenance by making the existence of the current leader unnoticed by other candidates, which may result in multiple nodes sending duplicate leader reports in the same period to the external world. However, the external world can always recognize duplicate reports through version numbers attached in the reports. Message loss can also prevent nodes on a group's outer boundaries from getting heartbeats. Consequently, these nodes may not become aware of the object and create spurious objects. However, Lightweight EnviroSuite allows members to disseminate heartbeats, which maintains a comparatively high heartbeat frequency and makes the possibility that a node fails to get any heartbeats very low. At the same time, objects attach their ages, which increase with the increase of finished task periods, to heartbeats. Object information from younger objects is discarded in favor of older ones. Therefore, spurious objects are eventually terminated due to their young ages.

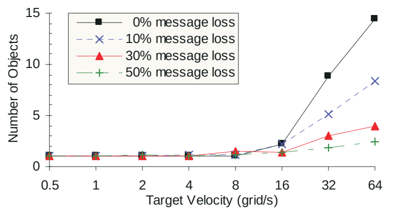

Fig. 15. Performance of object maintenance for varied message loss

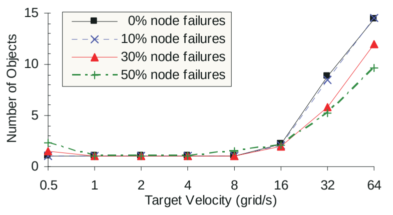

Fig. 16. Performance of object maintenance for varied node failures

Lightweight EnviroSuite is also robust to node failures. As stated earlier, it can go through leader deserts without losing object information or terminating task execution. In a similar way, it is able to overcome node deserts smaller than half an object resolution. Although we may temporarily lose track of the target inside a node desert, the target and its associated object can be picked up again in most cases after passing the node desert.

Experimental results confirm our conclusions as shown in Figure 15 and Figure 16. Lightweight EnviroSuite shows consistently good performance in object maintenance for targets with velocities up to 4 grid/s. However, after velocities exceed 8 grid/s, surprisingly bigger message loss results in fewer objects. This is because the number of objects is counted based on leader reports at the base station and a higher message loss results in fewer received leader reports, and thus fewer observed objects. For node failures up to 50%, Lightweight EnviroSuite exhibits comparable performance at velocities between 1 and 4 grid/s. However, when target velocities are as low as 0.5 grid/s, higher node failures do hurt performance. That is because higher node failures result in larger node or leader deserts. Slower targets may fail to cross the deserts before followers forget about the object.

4 System Evaluation

We integrated Lightweight EnviroSuite into an energy-efficient surveillance system, called *Vigilnet* [24], subsequently transitioned to the DIA. In December 2004, in the process of technology transition, we deployed 200 XSM motes running the Vigilnet system on sandy and grassy roads with a 3-way intersection and collected performance data in field tests. Figure 17 depicts the deployment of the system. Nodes are approximately deployed in a grid 10 meters apart, covering one 300-meter road and one 200-meter road. Each rectangular dot represents one XSM mote in the field. Several base-stations were deployed. Some nodes are missing in the GUI because they are turned off to emulate failures.

The XSM mote extends the MICA2 platform [25] by improved peripheral circuitry, new types of sensors and better enclosures. It communicates approximately 30 meters when deployed on grassy ground. The primary goal of the field test is to evaluate system ability to detect, classify and track one or multiple moving targets, which can be either SUVs, persons or persons carrying a ferrous object (suggestive of a weapon).

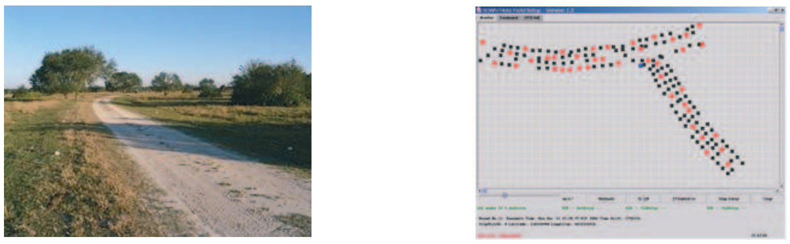
Fig. 17. System deployment

4.1 Overview of Vigilnet

Vigilnet is implemented on top of TinyOS. Figure 18 shows the layered architecture of Vigilnet. Components colored in dark grey are those implemented by Lightweight EnviroSuite.

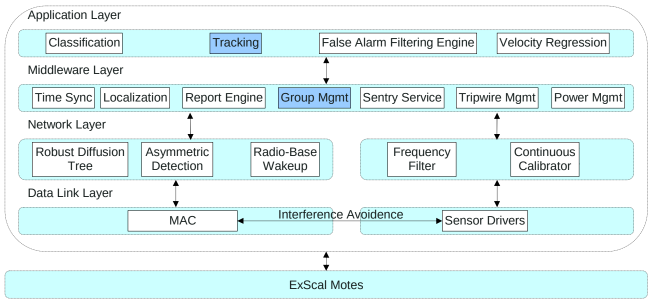

Fig. 18. System architecture of Vigilnet

Time synchronization (Time Sync), localization (Localization), and communication (MAC, Robust Diffusion Tree, Asymmetric Detection, and Report Engine) services constitute the lower-level components that are the basis for implementing higher-level services. Power management (Radio-Base Wakeup, Sentry Service, Power Mgmt, and Tripwire Mgmt), target classification (Sensor Drivers, Frequency-filter, Continuous Calibrator, Classification, and False Alarm Filtering Engine) and tracking (Group Mgmt, Tracking, and Velocity Regression) comprise main higher-level services. Target classification detects and classifies three types of targets with the help of collaborative group management provided by Lightweight EnviroSuite. Tracking components are responsible for estimating target positions and calculating target velocities.

Overall the system consists of 21,457 lines of source code, among which 2,884 are contributed by Lightweight EnviroSuite. The executable binary of Vigilnet occupies 85,926 bytes of code memory and 3,154 bytes of data memory, which can easily fit into XSMs equipped with 4KB data memory and 128KB code memory.

4.2 Tracking Performance Evaluation

Consistent with simulation, tracking modules in Vigilnet report to the base station estimations of target positions twice every second to provide sufficient data for false alarm processing and classification. A spanning-tree based routing [24] is used to disseminate such reports. The communication latency of these reports plays a critical role in achieving good tracking performance. Therefore, we suggest that when the system scales up to cover bigger fields, multiple base stations should be deployed. The false alarm filtering engine component executed in the base mote filters these reports and slows down the report rate to upper layers to once every 3 seconds due to bandwidth limitations. Target velocity calculation takes such reports as inputs.

Table 1 lists the comparison of tracking performance between Vigilnet equipped with the original EnviroSuite (measured at a previous field test conducted in August 2003) and Vigilnet equipped with Lightweight EnviroSuite. As is seen, without Lightweight EnviroSuite the maximum trackable velocity is about 5 to 10 mph, while the new Vigilnet system tracks targets up to 35 mph.

Table 1. Tracking performance comparison

Target vel.	Vigilnet with EnviroSuite	Vigilnet with Lightweight EnviroSuite
5 mph	successful	successful
10 mph	partially successful	successful
20 mph	failed	successful
30 mph	untested	successful
35 mph	untested	partially successful

Table 2. Tracking performance of Vigilnet with Lightweight EnviroSuite

Target type		Avg. tracking error	Std. dev. of tracking errors	Actual vel.	Calculated vel.
walking person		6.19 meter	3.28 meter	3±1 mph	2.9 mph
running person		6.67 meter	3.89 meter	7±1 mph	6.9 mph
vehicle		7.06 meter	3.98 meter	10±1 mph	10.5 mph
vehicle		5.91 meter	3.02 meter	20±1 mph	23.5 mph
two vehicles	1	5.58 meter	4.76 meter	10±1 mph	9.2 mph
	2	6.33 meter	3.52 meter	10±1 mph	9.9 mph

Note that, the success of tracking is an end-user metric measured by the accuracy of position and velocity calculations, which depend on several factors besides EnviroSuite group management protocols. A track is said to be successful only when the final calculated velocity within a 20% error. Due to the limited length of the field and the fixed report rate (once every 3 seconds), velocity calculation does not perform well when the velocity reaches 35 mph. However, the tracking performance of Lightweight EnviroSuite itself is actually better than the reported results for the integrated system.

Table 2 shows in more details the tracking performance of the new Vigilnet system. As seen, tracking errors are between 5.5 meters and 7.5 meters. These results were collected with 20% of the nodes randomly turned off to emulate failures. In all listed targets whose velocities vary from 3 mph to 20 mph, the maximum error of velocity calculation is less than 10%, which reflects the good tracking performance supplied by Lightweight EnviroSuite.

To give a more concrete view of the tracking performance of Lightweight EnviroSuite, Figure 19 shows the tracking trajectories for the following scenarios: (i) one vehicle drives across the field from left to right; (ii) two vehicles keep a distance of about 50 meters before they separate (the first one goes from left to right and turns right at the intersection and the second one goes from left to right). In the one-vehicle-tracking case, the rugged trajectory in the center of the horizontal road shows explicitly that existing node failures do affect tracking accuracy. The two-vehicle-tracking case proves the ability of Lightweight EnviroSuite to track multiple targets with the same sensory signatures as long as they keep a distance (50 meters) that is more than half an object resolution (set to 30 meters in the system). This was deemed sufficient by the client for operational use.

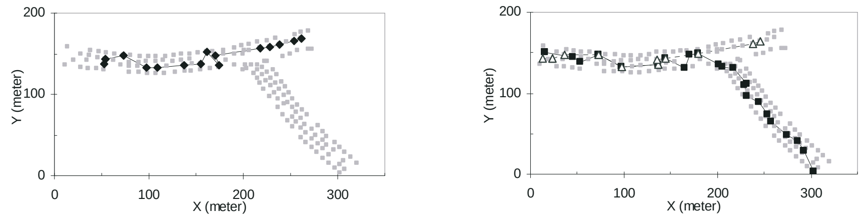

Fig. 19. Tracking trajectories for one vehicle and two vehicles

Field test results show that Lightweight EnviroSuite successfully improves the maximum trackable speed from near 10 mph to near 35 mph. Given our 10-meter-apart grid deployment and 20% node failures, tracking errors are still as small as about 6 meters and the maximum error in velocity calculation doesn't exceed 10%. These results from physically deployed systems validate that Lightweight EnviroSuite is practical, effective, and efficient on current hardware with limited communication and sensing capabilities. In this paper, we did not supply an experimental comparison between EnviroSuite and other high-level sensor network programming systems. This is, in part, due to the difficulty in porting application code across the different systems. If applications are re-implemented (as opposed to ported), it is hard to separate the effects of application-level implementation decisions from the inherent strengths and weaknesses of the underlying programming frameworks when interpretting performance comparison results.

5 Summary

Research on programming paradigms and frameworks in sensor networks has been very active in recent years. This paper presents our effort in improving an existing programming paradigm (EnviroSuite) into a more practical and efficient version (Lightweight EnviroSuite) that can be directly utilized to build practical large-scale systems with realistic requirements. The resulting version is shown to be efficient via experimental testing on a physically deployed large-scale surveillance system consisting of 200 XSM motes. This is one of the first attempts to use high-level sensor network programming languages in building and deploying real sensor network applications.

References

1. Boulis, A., Srivastava, M.B.: A framework for efficient and programmable sensor networks. In: OPENARCH '02. (2002)
2. Levis, P., Culler, D.: Mate: a virtual machine for tiny networked sensors. In: ASPLOS. (2002)
3. Li, S., Son, S.H., , Stankovic, J.: Event detection services using data service middleware in distributed sensor networks. In: IPSN '03. (2003)

4. Madden, S.R., Franklin, M.J., Hellerstein, J.M., , Hong, W.: The design of an acquisitional query processor for sensor networks. In: SIGMOD. (2003)
5. Yao, Y., Gehrke, J.E.: The cougar approach to in-network query processing in sensor networks. In: Sigmod Record, Volume 31, Number 3. (2002)
6. Welsh, M., Mainland, G.: Programming sensor networks using abstract regions. In: NSDI '04. (2004)
7. Newton, R., Welsh, M.: Region streams: functional macroprogramming for sensor networks. In: DMSN '04: Proceeedings of the 1st international workshop on Data management for sensor networks, New York, NY, USA, ACM Press (2004) 78–87
8. Liu, J., Liu, J., Reich, J., Cheung, P., , Zhao, F.: Distributed group management for track initiaition and maintenance in target localization applications (2004)
9. Whitehouse, K., Sharp, C., Brewer, E., Culler, D.: Hood: A neighborhood abstraction for sensor networks. In: MobiSYS '04. (2004)
10. Liu, J., Chu, M., Liu, J., Reich, J., Zhao, F.: State-centric programming for sensor-actuator network systems. In: IEEE Pervasive Computing. (2003)
11. XSM motes: (http://www.cast.cse.ohio-state.edu/exscal/index.php?page=main)
12. Stockman, H.: The active badge location system. In: Proceedings of the IRE. (1948) 1196–1204
13. Want, R., Hopper, A., Falcao, V., Gibbons, J.: The active badge location system (1992)
14. Liu, J., Chu, M., Liu, J., Reich, J., Zhao, F.: Distributed state representation for tracking problems in sensor networks. In: IPSN '04. (2004)
15. Aslam, J., Butler, Z., Constantin, F., Crespi, V., Cybenko, G., Rus, D.: Tracking a moving object with a binary sensor network. In: SenSys '03, New York, NY, USA, ACM Press (2003) 150–161
16. Shin, J., Guibas, L., Zhao, F.: A distributed algorithm for managing multi-target identities in wireless ad-hoc sensor networks. In: IPSN '03. (2003)
17. Zhang, W., Cao, G.: Optimizing tree reconfiguration for mobile target tracking in sensor networks. In: INFOCOM '04. (2004)
18. Blum, B., Nagaraddi, P., Wood, A., Abdelzaher, T., Son, S., Stankovic, J.: An entity maintenance and connection service for sensor networks. In: MobiSys '03. (2003)
19. Abdelzaher, T., Blum, B., Cao, Q., Evans, D., George, J., George, S., He, T., Luo, L., Son, S., Stoleru, R., Stankovic, J., Wood, A.: Envirotrack: Towards an environmental computing paradigm for distributed sensor networks. In: ICDCS '04. (2004)
20. Luo, L., Abdelzaher, T., He, T., Stankovic, J.A.: Envirosuite: An environmentally immersive programming framework for sensor networks. (In: Submitted to ACM Transactions on Embedded Computing Systems)
21. Karp, B.: Geographic Routing for Wireless Networks. PhD thesis, Harvard University (2000)
22. Levis, P., Lee, N., Welsh, M., , Culler, D.: Tossim: Accurate and scalable simulation of entire tinyos applications. In: SenSys '03. (2003)
23. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K.: System architecture directions for networked sensors. In: ASPLOS-IX, New York, NY, USA, ACM Press (2000) 93–104
24. He, T., Krishnamurthy, S., Stankovic, J.A., Abdelzaher, T., Luo, L., Stoleru, R., Yan, T., Gu, L., Hui, J., Krogh, B.: Energy-efficient surveillance system using wireless sensor networks. In: MobiSYS '04, New York, NY, USA, ACM Press (2004) 270–283
25. UC Berkeley. MICA motes: (http://www.tinyos.net/scoop/special/hardware/)