ACM International Conference on Pervasive Services, Sorrento, Italy, July 6-10, 2008, ACM Press, pp. 101-110.

# A Middleware Framework for Market-Based Actuator Coordination in Sensor and Actuator Networks

Joel W. Branch
IBM T.J. Watson Research Center
19 Skyline Drive
Hawthorne, NY, USA 10532
branchj@us.ibm.com

Lei Chen
Department of Computer Science
Rensselaer Polytechnic Institute
110 8th Street
Troy, NY, USA 12180
chenl6@cs.rpi.edu

Boleslaw K. Szymanski
Department of Computer
Rensselaer Polytechnic Institute
110 8th Street
Troy, NY, USA 12180
szymansk@cs.rpi.edu

## ABSTRACT
Sensor and actuator networks (SANETs) are a growing class of distributed systems combining sensors for environmental monitoring with actuators for reacting to environmental changes and controlling its dynamic processes. As SANETs evolve to accommodate complex large-scale deployments, there is an increasing probability that sensors' and actuators' attempted tasks may impact those of other sensors and actuators; for example by collectively exceeding the amount of available resources. Hence, there is a need for new tools that can autonomously coordinate the execution of SANETs for the benefit of the entire system. This need is especially important in case of actuators since they actually *affect* the environment. This paper advocates the use of market-based methods as the basis for distributed actuator coordination within the Sentire SANET framework. We demonstrate that auction-based distributed actuator coordination in an HVAC system leads to efficient, temporal and fair allocation of energy even when the desired temperature cannot be sustained in all locations simultaneously. We also discuss how this approach can be generalized to similar SANET resource allocation problems and what other challenges auction mechanism design faces when applied to SANETs.

## Categories and Subject Descriptors
D.2.11 [**Software Engineering**]: Software Architectures – *domain-specific architectures.*

F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General

## General Terms
Algorithms, Management, Design.

## Keywords
Sensor and Actuator Networks, Middleware, Auctions.



## 1. INTRODUCTION
Recent advancements in integrated microprocessor, storage, and wireless communication technology combined with associated cost reductions have catapulted the emergence of *wireless sensor networks* [2], [11]. Such networks are characterized by their use of small, unobtrusive nodes that detect environmental phenomena and wirelessly transmit the resultant data to a remote application or data store. In addition to sensor networks, a growing body of systems also requires the paired services of *actuators*, which are employed to enable prompt reaction to sensed events by controlling the environment or process under monitoring [1]. Such systems have often been referred to as *sensor and actuator networks*, or *SANETs*. An example includes an intelligent indoor sprinkler system that employs multimodal sensors to locate and collect vital information about fires in order to instruct sprinklers where to aim and discharge water or appropriate chemicals. Further examples are described in [19] and [25].

We emphasize that there is growing need for developing SANETs for large-scale distributed control and hence relying on a wealth of actuators and data sources of heterogeneous platforms (e.g., sensors, databases, and other event-generating endpoints). The feasibility and proliferation of such systems is increasing thanks to advancements in standards for network/device connectivity, service interoperability, and sensor data descriptions [21]. The magnitude and complexity of such systems will far exceed those of control systems in prior generations. A timely example includes a battlefield scenario in which a surveillance application uses a myriad of sensors of various modalities (e.g., acoustics, infrared imagery, seismic data) and domains of control (e.g., army, marines, allied countries) that determine the location and severity of threats over a large geographic area. Using these inputs, the application directs the most capable actuators (e.g., unmanned aerial drones, ground located, or even appropriately trained soldiers, etc.) to appropriately neutralize the threats.

Numerous challenges continue to impede the proliferation of large-scale SANETs; two notable ones are described below.

- *The design of SANET development tools*: It is highly inefficient for developers to cope with the lower level platform details of heterogeneous endpoints when designing SANETs. An intuitive reusable model and systematic process should be provided to address the high-level development of SANETs. Among other tasks, this will

require the development of novel programming abstractions that were previously unnecessary.

- *The design of actuator coordination techniques*: Integrating a large-scale distribution of networked actuators promotes decentralization, and hence increases the opportunity for actuator task interference, which can arise in multiple ways. First, the uncoordinated actions of physically nearby actuators can produce unwanted, or damaging environmental affects. Second, actuators often use a shared resource (e.g., energy or treatment supplies) to fulfill their tasks. Hence even distant actuators can still affect each other's ability to perform a task.

In response to the above challenges, our contribution is a novel SANET middleware framework, *Sentire* [5], [6], which provides extensible support for market-based distributed actuator coordination. Sentire is an end-to-end framework that supports the composition of extensible middleware for large-scale SANET-enabled systems in a high-level, platform-agnostic manner. To enable Sentire's support of actuator coordination, we propose the use of reusable framework components for aiding the implementation of market-based algorithms for distributed actuator coordination, which can also be modeled as a resource allocation problem. As opposed to traditional methods, market-based solutions provide a relatively simple and localized paradigm to incorporate the *value* of information to application objectives during coordination, thus resulting in feasible and meaningful allocation solutions. We focus on a configuration in which a large-scale SANET application is comprised of distributed Sentire middleware instantiations acting as self-interested agents to fulfill a global application goal. Each instantiation (or *agent*) is interfaced with sensors and actuators and they continuously bid against each other for common actuator resources (e.g., energy). Local bidding behavior is based on both adaptive and priority-based budgeting policies and environmental sensor data indicating an agent's status in fulfilling local actuation goals that support the global application's overall purpose. While in this work we largely address task interference relating to resource allocation, we later discuss our plans to extend Sentire's coordination facilities to the case where nearby actuator's actions may cause unwanted environmental effects due to overlapping treatment areas.

The rest of the paper proceeds as follows. Section 2 describes related works. Section 3 presents a background on the Sentire framework. Section 4 describes Sentire's market-based distributed actuator coordination techniques. Section 5 describes an evaluation of our proposal using a simulated case study and Section 6 presents the conclusion and future work.

## 2. RELATED WORKS

There have been various works addressing high-level SANET middleware. Garnet [23] emphasizes the utilization of sensor data streams as basic programming abstractions for composing sensor-enabled systems. Milan [15] places a large emphasis on maintaining applications' quality of service (QoS) requirements. This is accomplished via state-based variable graphs that Milan uses to match sensors to application variables (describing required data and QoS) needed for particular operating states. Kairos [13] employs a macroprogramming approach to managing sensors by enabling developers to translate global program behavior to local node behavior and provides programming abstractions for manipulating individual nodes, accessing their neighbors, and acquiring their data. More recent middleware contributions provide stronger support for actuators and coordination. Meditrina [9] describes a framework that manages control applications in ubiquitous computing environments. An activity manager decomposes application goals into device tasks and also reports anomalous control-loop behavior to the application while a task manager controls tradeoffs between QoS and network costs in task execution. Barbaŕan [3] describes an architecture that uses tuple channels to provide ordered delivery of coordination messages among sensors and actuators. The tuple channels can be dynamically interconnected to enable adaptive system architectures. Further examples of similar frameworks include [7] and [8].

We specifically propose using market-based mechanisms for distributed actuator coordination within a SANET middleware environment. One of the earlier uses of auctions in sensor networks was proposed in [20] where an auction-based actuator coordination solution is described. The proposed algorithm has a different goal than our work because it focuses on optimally selecting actuators to perform actions according to task completion time and energy usage. We do not address this problem. Hall *et al.* [14] propose a market-based approach to sensor network resource allocation. Here, sensors and network bandwidth are viewed as suppliers of services while applications are treated as their consumers. The value of the information provided by the supplier is based on metrics such as accuracy, timeliness, etc. A dynamic auction mechanism finds the optimal allocation of limited resources under various information requirements. A related work, MASM [24], employs a combinatorial search algorithm to find the optimal allocation for the problem described above. Both [14] and [24] focus on auction mechanism issues of applying market-based methods for sensor coordination in relation to resource allocation. In contrast, our paper concentrates on understanding how bidding strategies of coordination participants impact the quality of the solution obtained when the solution is time dependent and requires timesharing of resources.

## 3. THE SENTIRE FRAMEWORK: AN OVERVIEW

The Sentire framework facilitates the process of building extensible high-level application middleware that employs SANETs. Sentire addresses the process of building such a middleware, rather than addressing a specific instantiation of a middleware optimized for some specific purpose. Sentire's driving goal is to systematically guide the process of developing middleware by providing the following contributions:

- a high-level end-to-end framework consisting of extensible and reusable managers that (i) facilitate the flow of queries, commands, and data between applications and underlying SANETs, and (ii) partitions middleware development into logically related sub-tasks;
- programmable middleware components that shield developers away from intimate details of interacting with underlying SANETs and provide scalable control over underlying sensors and actuators;

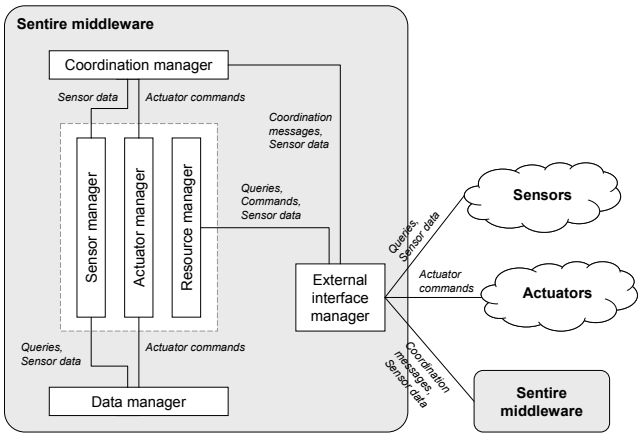


**Figure 1. Major components of the Sentire middleware framework.**

- a lightweight budgeting and auction-based policy for coordinating a set of actuators that are distributed among different Sentire instantiations and execute interfering local tasks.

The Sentire framework is largely defined by its set of extensible managers that are shown in Figure 1. The primary motivation behind this component architecture is to enable different aspects of SANET middleware to be developed and extended in an independent manner. Figure 1 also shows what data is passed both internal and external to the Sentire middleware. Sentire uses publish-subscribe messaging for facilitating communication between managers. Following is a brief description of the functionality of each manager.

The *External Interface Manager* serves as the interface for Sentire to communicate with sensors, actuators, and other related endpoints as well as other Sentire instantiations.

The *Resource Manager* associates SANET endpoints with Sentire, describes their features (e.g., data types and locations) and operating states (e.g., active and inactive), and controls applications' access to those endpoints via developer-defined policies that can accept or reject usage requests. The resource manager can also be extended by the developer to maintain additional application-specific information such as sensor/actuator energy levels, estimated network lifetime, link bandwidths, etc.

The *Sensor* and *Actuator Managers* implement the logical end-to-end binding aspects of the framework. The sensor manager provides lookup facilities for binding application queries with the most appropriate sensors and also provides a programming space for customized query decomposition (e.g., for querying a specific type or combination of sensors for satisfying a particular level of quality of information indicated in the query). The sensor manager also determines if similar queries can be matched with the same data source and if so, maintains a many-to-one query-to-sensor binding for efficiency. One-to-many bindings may be also maintained if necessary, such as in the case of a query requiring data averaged over a large geographic area. The actuator manager is similar to the sensor manager, although many-to-one bindings are not allowed because simultaneous control of single actuators is not supported.

The *Data Manager* provides a programming space for developer-defined application control logic. Supporting this, the data manager provides two major programming abstractions: *virtual sensor* and *virtual actuator interfaces*. These abstractions are used to retrieve sensor data, send actuator commands, and control the creation, deletion, and updating of bindings in the sensor and actuator managers according to their parameter values (e.g., sensor/actuator type, data rate, etc.).

Since this paper's main focus is actuator coordination, we refer the reader to [4] and [5] for more detailed descriptions of Sentire's managers, message format, and information flow. We note that a major aspect that these managers do not address is the translation of a query/command from Sentire's format to that of an underlying SANET endpoint. We expect this operation to occur at the endpoint (in an *adaptor* component) with the intent to both maintain one standard message/data format internal to Sentire and allow a more seamless transition if updates are made to the platforms' message/data format.

# 4. MARKET-BASED DISTRIBUTED ACTUATOR COORDINATION

We previously investigated distributed actuator coordination support for Sentire in [6]. However, this work involved a messaging protocol paired with a time-slotting mechanism for coordinating when interfering actuators could operate given sensor data values and expected treatment delays. Here, we describe a new market-based solution to actuator coordination which better promotes simplified local decision making and provides an entry point to use well-proven techniques from economic theory (e.g., determining *Pareto optimality* for equilibrium states) for performing coordination in the future. A more exhaustive justification for exploring market-based solutions in SANET-related middleware is descried in [12].

Sentire's market-based actuator coordination solution is supported by the *Coordination Manager (CM)*, which embodies the logic for distributed actuator coordination. Specifically, the CM helps manage the behavior of actuators that are distributed among different Sentire instantiations, but still use some resource from a shared entity as illustrated in Figure 2. This scenario can apply to large-scale control systems such as climate control systems, intelligent household and grid-based energy management systems, and intelligent agricultural/vineyard management systems. All of these systems are exemplary of large-scale SANET systems with global goals that can be mapped to the local behavior of distributed sub-entities. An example we use includes an office complex where each individual office hosts a Sentire middleware instantiation (hosted on some central control unit like a thermostat) that is charged with managing the climate for that office. Sentire would independently control a local heater unit (or units) using sensors deployed throughout the office. Coordination between Sentire instantiations of multiple offices becomes necessary if there is an energy budget imposed that prohibits all offices from simultaneously being heated (or cooled, as the need be) to their desired user preferences. This restriction might be imposed to keep energy costs low, reduce the probability of a grid brownout, or prolong the operation of a back-up generator supplying the building's energy.

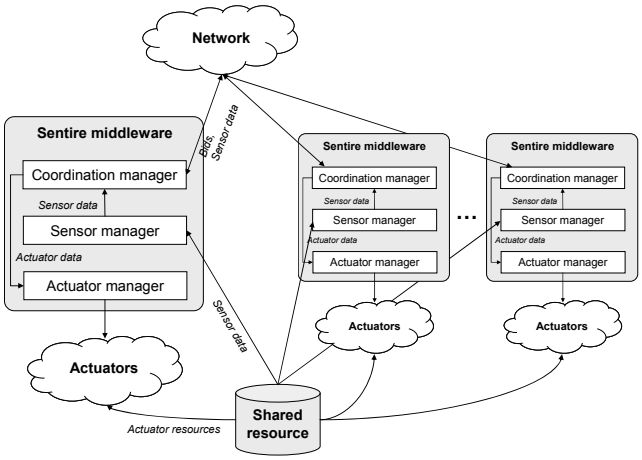


**Figure 2. Sentire's distributed actuator coordination configuration.**

## 4.1 The Sentire Coordination Manager

The CM facilitates actuator coordination using a distributed bidding scheme where each Sentire instantiation bids a percentage of an available budget for the use of a common resource. In general, the higher the bid is against other bidders, the greater fraction of resource that is allocated to the bidder. This scheme is beneficial for various reasons. One, it is simple, not requiring a great amount of computing power, yet is effective. Two, it is scalable, since no centralized application entity is required and only the message-passing overhead increases along with the number of participants, as with most any other coordination algorithm. We imagine that, in some cases, the CM will guide the entire behavior of the Sentire middleware. Such a scenario may include a winter setting in which there are frigid outdoor conditions and all office occupants want their offices constantly at a well-heated state. However, in other cases, coordination may only be needed periodically. We explain these details next.

The CM's operation is triggered by the data manager. A developer might determine that coordination is needed due to numerous rejections from the resource manager to perform actuation (because of lack of available resources). There is also a timer component in the data manager that can be set to trigger an alarm if a specified sensor data source's value does not exceed a threshold in a given amount of time, indicating that intended actuation is not sufficiently effective. Following such an event, the data manager can send a `BID` message to the CM including the bid amount (`BID.amount`) as well as the action type and location of the actuator (`BID.action` and `BID.location`[1], respectively). For simplicity, we currently assume that these parameters correspond to only one type of resource. However, extending this operation to a more general case is not difficult. When the CM receives the `BID` message, it will determine its allocation of resources according to the following equation:

$$\left(1 + \frac{1}{2^{n}} - \frac{1}{2^{a}}\right) \times \left(\frac{b_i^{2}}{\sum_{j=1}^{n} b_i^{2}}\right) \times te \ , \qquad (1)$$

where,

- $n$ = total number of known Sentire instantiations comprising the control system;
- $a$ = total number of known Sentire instantiations participating in coordination;
- $b_i$ = Sentire instantiation $i$'s current bid;
- $te$ = total amount made available to the entire system.

Both $n$ and $te$ are assumed to be known by all participants in the system. Figure 2 indicates that $te$ (or any other data associated with the shared resource) may be collected in the form of sensor data. We will shortly explain how all participants learn the value of $a$. Overall, Equation 1 will give each participant an equal allocation of resources, if bids are the same, no matter what the value of the bid is. The equation is also designed so that if there is only one participant that is bidding (i.e., $a$ = 1), then the entirety of the resources ($te$) will not be consumed by that sole individual, but only a fraction that asymptotically approaches 0.5 as $n$ grows. This should be acceptable since we expect the value of $te$ to be chosen such that $te$/2 should be enough to satisfy the requirements of any one actuator acting alone in the system. Furthermore, this fraction can be acquired using a very low bid. Overall, this is beneficial for those systems that may always operate in a state of coordination. It also implements the often postulated requirement that the higher the demand for the resources is (as expressed by the number of similar bids for energy), the higher the price of the resource unit is. Indeed, according to Equation 1 the same bid buys less and less resources as the number of bidders with the same bid increases.

Continuing, when the CM receives the bid from the data manager, it will send a `BID` message to all other participants (i.e., Sentire instantiations) of the system. We do not specify how this message is propagated, but assume that either the CM knows the network addresses of all other Sentire instantiations, or a multicast or flooding protocol has been implemented in the network to enable distributed message propagation.

When the CM receives a `BID` message from an external Sentire instantiation, it will decide to join in coordination only if it knows that its own host is currently attempting to actuate using the same action type as described in `BID.action`. This information is determined simply by querying the actuator manager. If the CM decides to participate in coordination, it will send a `BID.message` to all other members of the system as well. If the CM decides not to participate, it will simply transmit a `BID` with null values. Using this process, each member of the system participating in bidding will learn the value of $a$ as well as the total of everyone's bids and, hence, be able to use Equation 1 to determine their resource allocations. Furthermore, this process makes the decision-making logic of this system completely distributed. While this describes how the CM manager operates, we do not intend to indicate that bidding is strictly a continuous

[1] The `BID.location` field is meant to be used in cases where actuator coordination is needed for reducing interference due to overlapping treatment regions; hence it is not used in this paper.

process. The frequency at which bidding occurs is left up to the developer to decide. For instance, in the case study described in Section V, we describe several possible ways to control bidding. However, we assume that the frequency with which resource allocation needs to change is relatively low, so we can further assume that the communication bandwidth necessary for broadcasting the bids is a small fraction of available bandwidth. Hence, the overhead of communication in this solution is negligible.

A caveat of the above process is that symmetric bidding strategies can lead to continuously identical bids and will only result in each participant gaining an equal share of resources. If this occurs, it is possible that no actuator will ever have an opportunity to fulfill its local goal and budgets will be wasted. Hence, as a rule of thumb, there should be some factors that differentiate when or how long a Sentire instantiation is able to acquire all the resources it needs so that other instantiations will have the chance to meet their respective goals. It is non-trivial, and perhaps not appropriate, for the CM to enforce in a generic sense how bidding is differentiated because this is a very application-oriented feature.

The amount of a bid or when a bid is placed may be a factor of any number of features (e.g., current task, priority, time elapsed since last time the actuator's goal was reached, etc.) that are specific to the Sentire instantiation. This information is straightforward for the CM to obtain. However, our investigation of example applications reveals that some knowledge of other participants' sensor data could be useful in guiding bidding behavior. We show an example of this in Section V. The CM supports this by being able to collect the sensor data of other Sentire instantiations, as illustrated in Figure 1. When a `BID` message is sent, it can include any sensor data that the developer chooses to place in the message. One might suggest more flexible ways of sharing sensor data, such as allowing the CM to query the sensor data of other Sentire instantiations. However, the requester must have intimate knowledge of what sensor data to collect. This is non-trivial since only certain data guides an actuator's tasks. For instance, knowing what *type* of sensor data to collect is fine, but knowing the *location*, which strongly guides an actuator's behavior, is difficult. Hence, allowing the bidder the opportunity to include whatever sensor data he sees fit for other bidders to make informed decisions is the most straightforward option.

Furthermore, the implementation of this auction mechanism can range from a fully distributed one to fully centralized one. In fully distributed solution, each CM acts as a local auctioneer and therefore needs to have full knowledge of all bids and activities. Hence, this is the most communication intensive but also most reliable implementation in which failures of some CMs would not impact others. A fully centralized solution has a single auctioneer (one of the CM's) that collects all the bids and returns resource allocations. This is the least communication intensive. However, this is also the least reliable one because a failure of an auctioneer will bring the entire system down. A middle ground solution can use a certain number of auctioneers (co-located with the selected CMs) which then need to communicate with each other, but the system can survive failure of some of them. The choice of the implementation is application architecture dependent, so it will not be discussed here. However, such a choice will dictate the communication overhead of the solution, therefore we also do not measure or discuss this overhead as it will drastically change with the type of implementation architecture.

# 5. CASE STUDY

In order to demonstrate the benefits of Sentire's actuator coordination mechanism, we have developed a usage scenario involving a simulated HVAC control system. We believe that this example easily generalizes to other applications of sensor and actuator systems and therefore can serve as a paradigm of application of market-based mechanisms for distributed coordination.

## 5.1 HVAC Model

We used the SimJava simulation framework [22] to implement a model of an office complex's climate change behavior. The complex consisted of four separate rooms with independent climate, or heat loss, properties. In the future, we plan to use a larger number of controlled entities in order to formally test the scalability of Sentire's performance. However, this current configuration serves as a sufficient proof of concept. To calculate the heat loss of a room (defined as the rate at which heat leaves a room), we used the following set of equations:

$$heatloss = heatloss_{walls} + heatloss_{windows} + heatloss_{\inf iltration}, \quad (2)$$

$$heatloss_{walls} = \Delta t \times \left( \frac{a_{outsidewall}}{r_{outsidewall}} + \frac{a_{insidewall}}{r_{insidewall}} + \frac{a_{ceiling}}{r_{ceiling}} \right), \quad (3)$$

$$heatloss_{windows} = \Delta t \times a_{windows} \times tf_{windows}, \quad (4)$$

$$heatloss_{\inf iltration} = \Delta t \times \rho \times ER \times c \times v_{room}, \quad (5)$$

where the variable definitions and values used for this case study are as follows:

- $\Delta t$ = indoor temperature minus outdoor temperature (°F);
- $a_x$ = the area of $x$ with $x$ being outside walls, inside walls, ceilings and windows (68 ft$^2$, 564 ft$^2$, 140 ft$^2$ and 16 ft$^2$ respectively);
- $r_x$ = the thermal resistance of $x$ with $x$ being outside walls, inside walls and ceilings (16.97 hr·ft$^2$·°F/BTU, 0.45 hr·ft$^2$·°F/BTU and 49 hr·ft$^2$·°F/BTU respectively);
- $tf_{windows}$ = the transmission factor of the windows (1.13 BTU/(hr·ft$^2$·°F));
- $\rho$ = air density (0.0741 lb/ft$^3$);
- $ER$ = the exchange rate of air (0.4 (air change)/hr);
- $c$ = the specific heat of air (0.24 BTU/(lb·°F));
- $v_{room}$ = the volume of a room (1680 ft$^3$).

Heat loss determines the rate of heat a heater needs to generate to sustain a specific indoor temperature. So, if a room experiences a

heat loss rate of $x$ when the indoor temperature is 70°F and the outdoor temperature is 30°F, then a heater needs enough power to generate $x$ BTU/hr to keep the room's temperature at 70°F.

It should be noted that the computation of the heat loss is needed only for simulation. In fact, in real life, it is unrealistic to assume that the amount of energy needed to heat the room to the required temperature can be so easily computed. Factors such as sun exposure of the room, humidity, rain, number of people in the room, and time of the day will influence such a computation. Hence, in real life the system will simply react to the changes in the room temperature according to the current bidding strategy. The advantage of using auctions in this situation is that neither the bidders nor the controller need to know how much energy is needed to raise the temperature in the room by the given amount. Rather, the auction mechanism allows for dynamic response of the HVAC system to the current and often changing environmental conditions of each room.

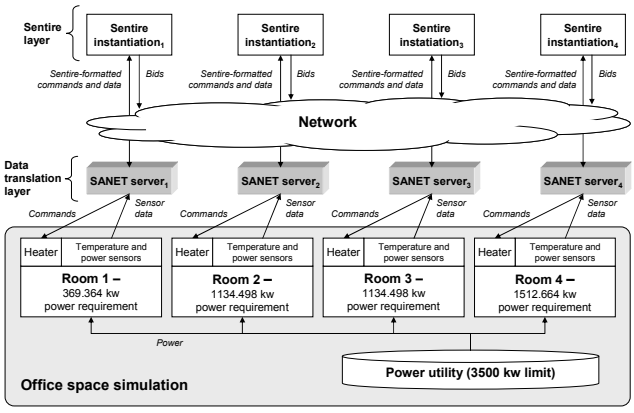


**Figure 3. Case study illustration.**

## 5.2 System Architecture

The architecture for this example is depicted in Figure 3. In this simulation, a separate Sentire middleware instantiation was associated with each room, where sensor data consisted of temperature readings and actuator commands included changing the power setting (in kw) of the room's heater. The goal of the system is to make effective usage of limited and shared power resources to maintain each room's individual temperature as close to the desired temperature with minimum usage of the energy. The value *te* (represented as *power utility* in Figure 3) was set to 3500kw, which was well below the power needed for all rooms to simultaneously sustain their temperature goals, forcing the need for coordination. Each room's temperature goal was set to 70°F, however, not all rooms required the same amount of power to reach this temperature. To enforce this, we gave each room its own outdoor temperature for purpose of simulation. We set room 1, 2, 3 and 4's outdoor temperature to 60°F, 40°F, 40°F, and 30°F respectively, modeling each room's different exposure to the Sun. Each room's power requirement in shown in Figure 3.

The activities of the Sentire CM in this example were as follows. First, we configured the system in such a way that the CM always drove the operation of the Sentire middleware. Hence, there were no messages sent from the data manager. We chose this setup because the purpose of this example is to focus on actuator coordination and also to provide consistent operation of the CM which allowed for more meaningful data plots (i.e., without anomalous data points representing the absence of any coordinated behavior). As mentioned previously, each `BID` message included the bid value as well as the pertinent sensor data value (`BID.sensor_value`). We extended the CMs so that each had a maximum budget of 50. When the budget was depleted, we used sensor data to help differentiate (based on priority) when the rooms' Sentire instantiations were allowed to bid by controlling the rate at which each room increased its budget. CMs were only allowed to bid for power after their budgets increased back to the full initial amount.

By design, there is not enough energy to consistently maintain the desired temperature in all rooms. Hence, we aim at finding a solution in which the temperature oscillates around a certain equilibrium temperature, so at least part of the time the room is at or above the desired temperature and the lower temperature may be easily tolerated if there are no occupants or activities in the room. Accordingly, we do not allow the use of the partially depleted budget until it is reset to its initial value. It is beyond the scope of this paper to consider stable temperature solutions.

## 5.3 Budget Management and Bidding Strategies

Each CM used the following rules to dictate the rate at which its budget increased:

- IF (<room's temperature> is within 10% of <mean temperature for all rooms>) AND (room's heater is turned off)
  THEN increase budget at a rate of [2+0.001·(*te*− <current power usage for all rooms>) every 10 seconds;
- IF (<room's temperature> is greater than 10% of <mean temperature for all rooms>) AND (room's heater is turned off)
  THEN increase budget at a rate of 1 every 10 seconds;
- IF (<room's temperature> is less than 10% of <mean temperature for all rooms>) AND (room's heater is turned off)
  THEN increase budget at a rate of [3+0.001·(*te*− <current power usage for all rooms>) every 10 seconds.

These rules were designed in accordance with the dynamic power requirement of each room to accomplish its individual heating goal. We assume that the current power usage for all rooms can easily be obtained as sensor data from the shared power source. Another way to learn this data is for a room's CM to include it in its `BID` message.

To perform the evaluation, we collected some application performance metrics using several different CM bidding strategies. These strategies are described below:

S1. IF <room's temperature> falls below 70°F THEN bid the entire budget (50) and use the full amount of the allotted power to run the heater;

S2. IF <room's temperature> falls below 70°F THEN bid the entire budget (again using the full amount of the allotted power)

IF <room's temperature> surpasses 75°F
THEN deactivate the heater and start increasing the budget according to the previously described rules;

S3. All CM's begin by bidding 1 unit of budget;
IF (the previous bid yielded a temperature gain) AND (room's temperature is below 70°F) AND (some budget remains)
THEN keep the previous bid invested;
IF (the previous bid yielded a temperature loss) AND (room's temperature is below 70°F) AND (some budget remains)
THEN bid an additional unit of budget;
IF (the room's temperature is greater than 75°F) AND (heater is on)
THEN turn off heater and reduce bid to 0 to save remaining budget.

S4. Same as S3, only bids of 3 units are used;

S5. Same as S3, only a bid of 3 units is used if 3 consecutive bids of 1 resulted in continuous temperature losses.

The CM turns the heater off any time no budget remains. Also, any time a new bid is received, the CM will adjust its power allocation via a re-evaluation of Equation 1. Strategies S1 and S2 use timers initialized to 500 seconds to control how long a heater is allowed to stay on while the room's temperature is above 70°F. When the timer expires, the CM will turn off the heater, set its budget to 0, and start to increase its budget according to the previously described rules. Again, here bids are made only when the budget is at its maximum value. This helps to give all CMs the opportunity to gain adequate power to heat their respective rooms past the target temperature. For strategies S3-S5, bidding occurs every 10 seconds. All simulations were run for 4000s.

These strategies describe different approaches to bidding, where Strategy S1 clearly is the simplest but will lead to the highest oscillations in the temperature as all budget is depleted once the heating starts. Strategy S2 tries to avoid this pitfall by limiting upper temperature reached during heating. Strategy S3 is adaptive, increasing its bids until the desired outcome is achieved while Strategy S4 is a more aggressive variant of Strategy S3. Strategy S5 makes aggressive bid increases but less frequently than Strategy S4. Overall, these strategies measure the sensitivity of the mechanism to variations in bidding.

For each strategy tested, we plotted and compared:

- temperature per room (°F), and
- average and cumulative average percent of power savings.

The average percent of power savings was calculated using the average power usage of all rooms and the total power usage needed for all rooms to constantly sustain a temperature of 70°F.

## 5.4 Results

The results shown in Figure 4 illustrate the temperatures the rooms reached as a result of strategies S1-S5. We clearly see the disadvantages of employing strategies S1 and S2, as Figure 4(a) and 4(b) show that room 1 is the most frequently heated above 70°F while room 4 largely suffers. This shows that while room 4 consumes more energy than room 1 for strategies S1 and S2, it is not enough to serve room 4's temperature needs. However, the remaining strategies, whose behavior is shown in Figure 4(c) – Figure 4(e), yield a much larger degree of fairness in the system,

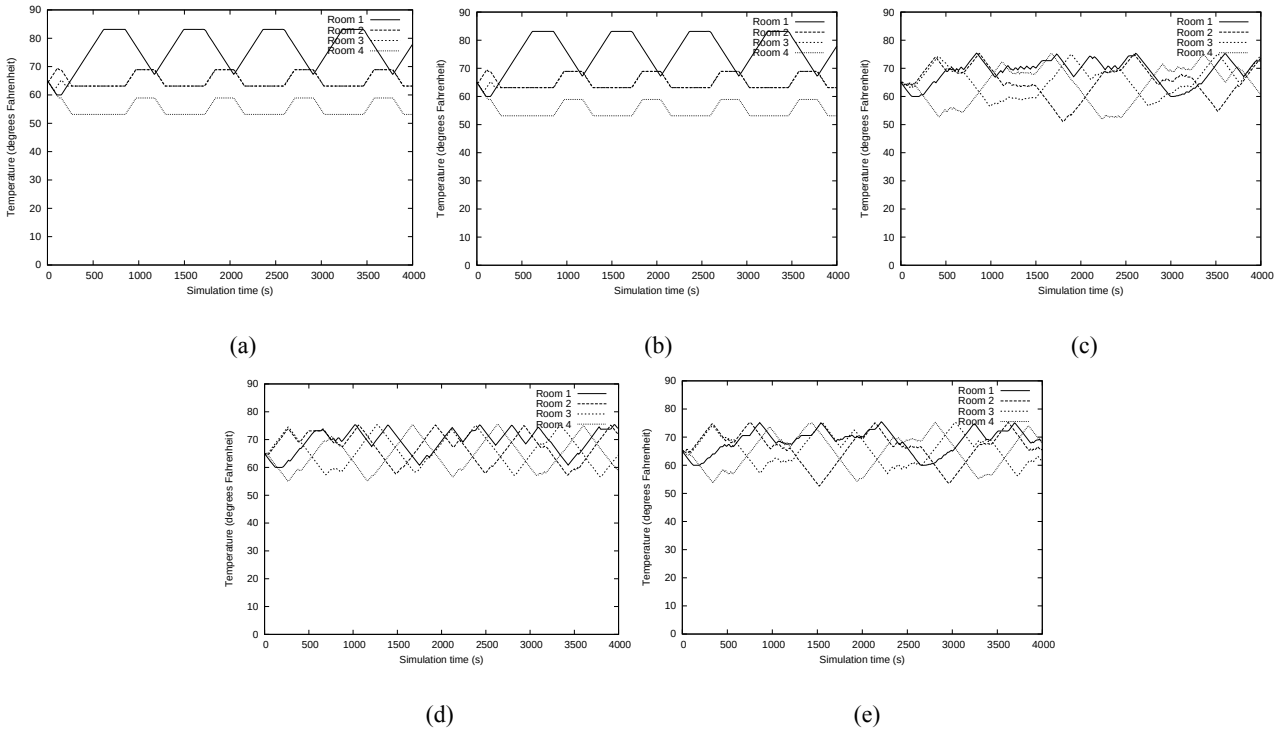


**Figure 4. Temperature reached per room for bidding strategies 1-5 (shown in subfigures a-e respectively).**

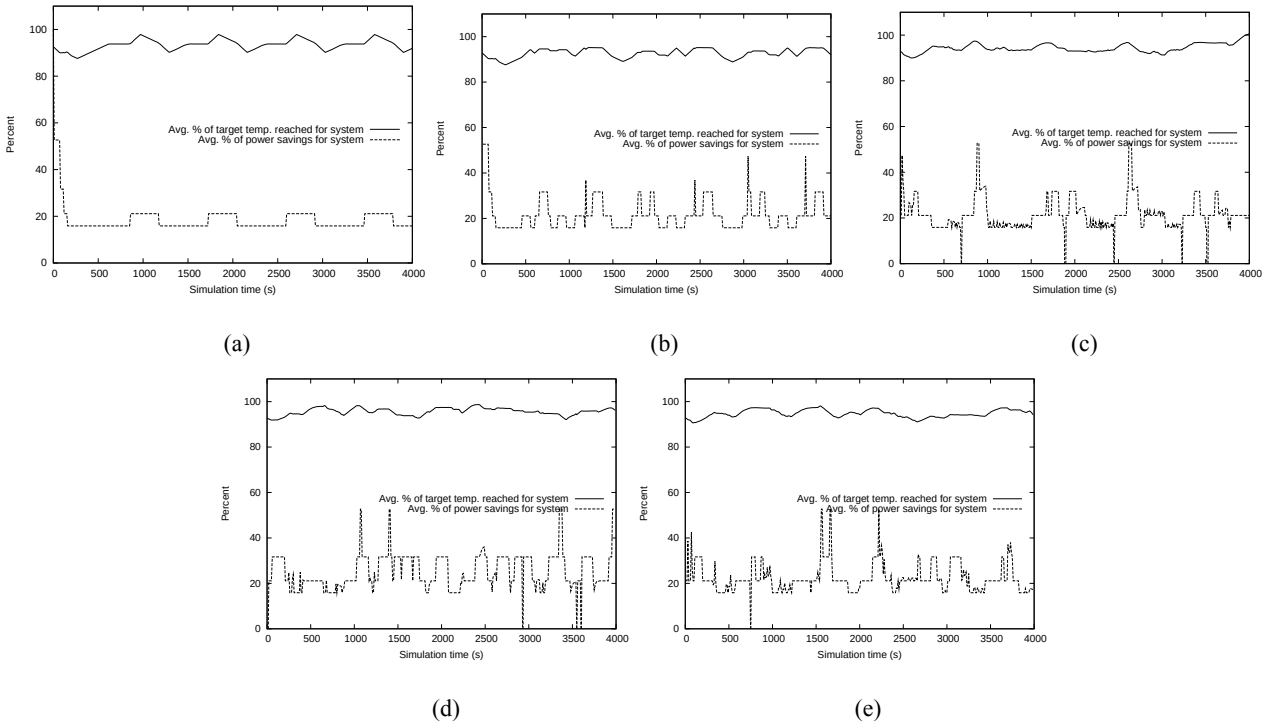


**Figure 5. Average percent of temperature goal reached and power saved for the entire HVAC system for strategies 1-5 (shown in subfigures a-e respectively).**

as all rooms are periodically heated to at least 70°F. A particular advantage of strategy S4, which uses a more aggressive bidding strategy (i.e., bidding 3 units as opposed to 1) is that no room's temperature drops far below about 55°F, as it does with strategies S3 and S5.

Figure 5 shows the power savings that a strategy is able to achieve in relation to the percent of target temperature reached. All strategies help the entire system reach the target temperature roughly 90% of the time. However, in pairing these plots with those shown in Figure 4, we know that this average does not reflect the starvation of room 4 for strategies S1 and S2. Strategy S1's behavior, depicted in Figure 5(a), is not as efficient as the other strategies since not only does it yield poor temperature performance (according to previous plots), but it only saves about 20% of the system's power consumption on average. The other strategies do much better. Strategy S2 periodically induces larger power savings again because the heater cuts off at a temperature beyond 75°F. Strategy S4, shown in Figure 5(d), appears to yield the greatest amount of power savings throughout the simulation. Figure 6 shows the cumulative values for the data shown in Figure 5. Inspecting this figure in conjunction with all the previous ones confirms the strong performance of Strategy S4. This strategy helps room 4 avoid starvation of resources, yields the second greatest overall percent of target temperature reached for the system (shown as percent not reached in Figure 6), and yields the largest percent of system power savings. Strategy S5 is also rather efficient helping the system reach an even larger percentage of its temperature goal and yielding nearly 25% power savings. This shows that a developer might choose either strategy

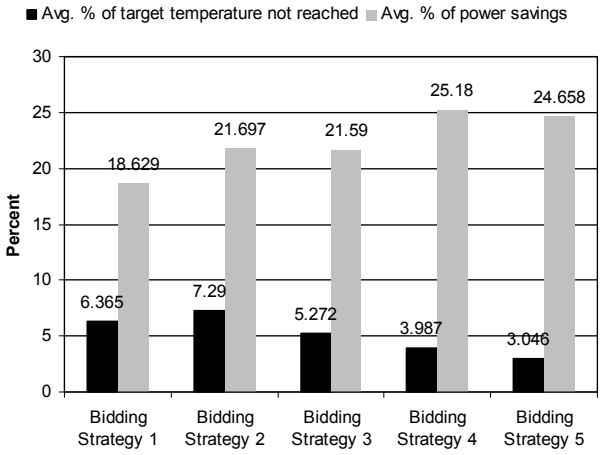


**Figure 6. Cumulative average percent of temperature goal not reached at power saved for each strategy.**

S4 or S5 depending on the goal of system being either power savings or actuation priority.

# 6. CONCLUSIONS AND FUTURE WORKS

In this paper, we presented a market-based solution to distributed actuator coordination using the Sentire SANET middleware framework. We showed how the Sentire framework could be used to easily provide coordinated actuator behavior using a distributed

HVAC control scenario. We conclude by outlining our future research agenda to extend this work.

## 6.1 Future Work on Auction Mechanisms

First, we intend to investigate which auction mechanism designs are stable and optimal as well as which ones possess other desirable properties in SANET environments. Such designs must accommodate the unique properties of SANET-related tasks with the most critical among them being:

- the interrelation of different tasks executed by sensors or actuators with overlapping ranges of influence;
- the necessity of SANETs to integrate several tasks into one composite task that, at least, partially satisfies them all;
- the use of complex, location and time-dependent cost functions of resource usage and application performance.

These properties must be accommodated while conducting repeated auctions for SANET resource allocation. We briefly discuss challenges regarding some of these items here.

The challenge of non-standard cost functions is partially addressed in [10] in which market-based mechanisms are used to solve the task allocation problem in which suppliers have limited capacity and a cost function is composed of a fixed overhead cost and a constant marginal cost. This model well approximates the SANET environment in which the fixed cost is that of network deployment and the marginal cost represents the energy needed to provide a service per time unit. Here, agents have incentives to overstate their capacity to receive higher shares of services when traditional auction mechanisms are used. In response, the authors propose an extended Vickrey-Clarke-Groves mechanism that enforces separate allocation and payment phases and a penalty scheme to motivate truthful reports of capacities and exhibits strategy-proof operation, efficiency, and optimal allocation. A decentralized mechanism, derived from the continuous double auction method, is also proposed, but does not guarantee full efficiency.

The challenge of repeated auctions for resource allocation is addressed in [16]-[18]. The authors show that applying traditional auction mechanisms to some service-oriented marketplaces may result in a *bid drop problem*. This problem is solved for homogenous and heterogeneous markets using either the first price sealed bid, or the second price sealed bid or even combinatorial auctions. The shared approach in all three cases is to award bidder's participation, so as to maintain the price competition and avoid the bidder drop problem during recurring auctions. We plan to integrate techniques inspired from [10] for dealing with non-linear utility and cost functions with the bidder drop avoidance mechanisms introduced in [16]-[18]. One approach to such integration will rely on providing an appropriate penalty scheme to avoid overbidding and to reward bidders for participation in the auction.

## 6.2 Application to Sensor Network Management

In addition to the HVAC scenario, we plan to investigate multi-object tracking by a military sensor network, which is a problem with a similar coordination structure in which each tracking mission requires a certain constant bandwidth to continuously collect information about its tracked object(s). When congestion arises somewhere on paths from the reporting sensors to the base station, the problem of the optimal bandwidth allocation must be solved. Similar to the HVAC example, the optimal resource allocation must be temporal and agents must be prevented from starving continuously of resources. Hence, the solution that we applied in the HVAC example of imposing a reward for the time that the agent starves and payment for the time the participant has the resources allocated applies to this problem as well. The reward reflects the fact that each mission that is denied bandwidth has the quality of information about its tracked object(s) decaying with time during which the updates are not arriving. This problem is further complicated by the face that different missions may have different and dynamically changing priorities that are independent of their level of resource starvation. We conjecture that the fundamental methodology developed for the HVAC example will apply to multi-object tracking as well. However, extensions are needed to address other application specific properties, such as accommodating multi-dimensional mission priorities.

## Acknowledgement

Research was sponsored by US Army Research Laboratory and the UK Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defence, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

## 7. REFERENCES

[1] I.F. Akyildiz and I.H. Kasimoglu. 2004. Wireless sensor and actor networks: research challenges. Ad Hoc Networks Journal. 2, 4, 351-367.

[2] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. 2002. Wireless sensor networks: a survey. Computer Networks Journal. 38, 4, 393-422.

[3] J. Barbarán, M. Díaz, I. Esteve, D. Garrido, L. Llopis, B. Rubio, and J. Troya. 2007. Programming wireless sensor and actor networks with TC-WSANs. Proceedings of 2007 IEEE International Conference on Pervasive Services (ICPS'07). 196-203.

[4] J. Branch. 2007. Opportunistic routing and middleware composition for sensor and actuator networks. Doctoral Thesis. Rensselaer Polytechnic Institute.

[5] J. Branch, J. Davis, D. Sow, and C. Bisdikian. 2005. Sentire: a framework for building middleware for sensor and actuator networks. Proceedings of 2005 IEEE International Conference on Pervasive Computing and Communications (PERCOM'05) Workshops. 396-400.

[6] J. Branch, B.K. Szymanski, C. Bisdikian, N. Cohen, J. Davis, M. Ebling, and D. Sow. 2006. Towards middleware components for distributed actuator coordination.

Proceedings of 3rd IEEE Workshop on Embedded Networked Sensors (EmNets'06).

[7] I. Chatzigiannakis, A. Kinalis, and S. Nikoletseas. 2006. Priority based adaptive coordination of wireless sensors and actors. Proceedings of 2nd ACM International Workshop on QoS and Security for Wireless and Mobile Networks (Q2SWinet'06). 37-44.

[8] P. Costa, L. Mottola, A.L. Murphy, and G.P. Picco. 2006. TeenyLime: transiently shared tuple space middleware for wireless sensor networks. Proceedings of the International Middleware Workshop on Middleware for Sensor Networks, co-located with Middleware'06.

[9] R. Crepaldi, A. Harris, R. Kooper, R. Kravets, G. Maselli, C. Petrioli, and M. Zori. 2007. Managing heterogeneous sensors and actuators in ubiquitous computing environments. Proceedings of ACM Workshop on Sensor and Actor Networks. 35-42.

[10] R.K. Dash, P. Vytelingum, A. Rogers, E. David, and N.R. Jennings. 2007. Market-based task allocation mechanisms for limited-capacity suppliers. IEEE Transactions on Systems, Man, and Cybernetics, 37, 3, 391-405.

[11] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. 2001. Instrumenting the world with wireless sensor networks. Proceedings of the International Conference on Acoustics, Speech and Signal Processing.

[12] B. Gerkey and M. Matarić. 2002. A market-based formulation of sensor-actuator network coordination. Proceedings of the AAAI Symposium on Intelligent Embedded and Distributed Systems. 21-26.

[13] R. Gummadi, O. Gnawali, and R. Govindan. 2005. Macro-programming wireless sensor networks using Kairos. Proceedings of the International Conference on Distributed Computing in Sensor Systems (DCOSS). 126-140.

[14] D. Hall and T. Mullen. 2007. On the utilization of market-based auction techniques for dynamic resource allocation in distributed sensor network systems. Proceedings of the 10th International Conference on Information Fusion.

[15] W.B. Heinzelman, A. Murphy, H. Carvalho, and M. Perillo. 2004. Middleware to support sensor network applications. IEEE Network. 18, 1, 6-14.

[16] J.-S. Lee and B.K. Szymanski. 2005. A novel auction mechanism for selling time-sensitive e-services. Proceedings of the 7th International IEEE Conference on E-Commerce Technology (CEC'05). 75-82.

[17] J.-S. Lee and B.K. Szymanski. 2005. Stabilizing markets via a novel auction based pricing mechanism for short-term contracts for network services. Proceedings of the 9th IFIP/IEEE International Symposium on Integrated Network Management. 367-380.

[18] J.-S. Lee and B.K. Szymanski. 2006. Auctions as a dynamic pricing mechanism for e-services. Service Enterprise Integration. Cheng Hsu, Ed. Kluwer, New York, NY, 131-156.

[19] S. Li. 2006. Wireless sensor actuator network for light monitoring and control application. Proceedings of IEEE Consumer Communications and Networking Conference. 974-978.

[20] T. Melodia, D. Pompili, V.C. Gungor, and I.F. Akyildiz. 2005. A distributed coordination framework for wireless sensor and actor networks. Proceedings of 2005 ACM International Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc'05). 99-110.

[21] SensorML. http://vast.uah.edu/SensorML/.

[22] SimJava. http://www.dcs.ed.ac.uk/home/hase/simjava/.

[23] L. St. Ville and P. Dickman. 2003. Garnet: a middleware architecture for distributing data streams originating in wireless sensor networks. Proceedings of 2003 IEEE International Conference on Distributed Computing Systems (ICDCS'03). 235-241.

[24] A. Viswanath, T. Mullen, D. Hall, and A. Garga. 2005. MASM: a market architecture for sensor management in distributed sensor networks. Proceedings of the SPIE Defense and Security Symposium. 281-289.

[25] T. Wark, C. Crossman, W. Hu, Y. Guo, P. Valencia, P. Sikka, P. Corke, C. Lee, J. Henshall, K. Prayaga, J. O'Grady, M. Reed, and A. Fisher. 2007. The design and evaluation of a mobile sensor/actuator network for autonomous animal control. Proceedings of 2007 ACM International Conference on Information Processing in Sensor Networks (IPSN'07). 206-215.