Multi-Application Deployment in Shared Sensor Networks based on Quality of Monitoring

Sangeeta Bhattacharya, Abusayeed Saifullah, Chenyang Lu, and Gruia-Catalin Roman
Department of Computer Science and Engineering
Washington University in St. Louis
{sangbhat, saifullaha, lu, roman}@cse.wustl.edu

***Abstract*—Wireless sensor networks are evolving from dedicated application-specific platforms to integrated infrastructure shared by multiple applications. Shared sensor networks offer inherent advantages in terms of flexibility and cost since they allow dynamic resource sharing and allocation among multiple applications. Such shared systems face the critical need for allocation of nodes to contending applications to enhance the overall Quality of Monitoring (QoM) under resource constraints. To address this need, this paper presents *Utility-based Multi-application Allocation and Deployment Environment (UMADE)*, an integrated application deployment system for shared sensor networks. In sharp contrast to traditional approaches that allocate applications based on cyber metrics (e.g., computing resource utilization), UMADE adopts a cyber-physical system approach that dynamically allocates nodes to applications based on their QoM of the physical phenomena. The key novelty of UMADE is that it is designed to deal with the *inter-node QoM dependencies* typical in cyber-physical applications. Furthermore, UMADE provides an integrated system solution that supports the end-to-end process of (1) QoM specification for applications, (2) QoM-aware application allocation, (3) application deployment over multi-hop wireless networks, and (4) adaptive reallocation of applications in response to network dynamics. UMADE has been implemented on TinyOS and Agilla virtual machine for Telos motes. The feasibility and efficacy of UMADE have been demonstrated on a 28-node wireless sensor network testbed in the context of building automation applications.**

I. Introduction

While wireless sensor networks (WSNs) have been traditionally tasked with single applications, recent years have witnessed the emergence of shared sensor networks as integrated cyber-physical systems infrastructure for a multitude of applications. Examples of shared sensor networks include recent deployment of urban sensing systems [1], [2], building automation [3], and integrated environmental monitoring [4]. For example, a smart building may employ an integrated WSN to support multiple applications including temperature and humidity monitoring, security alarms, light control, and structural health monitoring. Compared to a WSN dedicated to a single application, a shared WSN can significantly reduce the system cost by allowing multiple applications to share nodes and the network. It can also enhance system flexibility by dynamically allocating nodes to different applications in response to environmental changes and user requirements.

As WSNs evolve from application-specific platforms to shared cyber-physical systems infrastructure, they face the new challenge to allocate nodes and resources to contending applications subject to the resource constraints of sensor nodes. In contrast to application allocation in traditional computing systems only concerned with cyber performance metrics (e.g., latency and throughput), a shared sensor network must allocate applications based on their *Quality of Monitoring (QoM)* of physical phenomena due to the close coupling of the cyber and physical aspects of distributed sensing applications. The QoM of a distributed sensing application usually depends on the set of nodes allocated to it. Moreover, the measurements of different sensors are often highly correlated [5] resulting in *inter-node dependency*, i.e., the QoM contributed by a node to an application is dependent on the other nodes allocated to the same application. For example, intelligent air conditioning control requires an accurate estimation of the temperature distribution in the building based on the measurement of a finite set of nodes. Allocating a new node whose reading is highly correlated to that of one already assigned to the application, will not significantly increase the information about the temperature distribution. Such inter-node QoM dependencies specific to cyber-physical interactions introduce unique challenges to application allocation that has not been addressed by existing approaches to application allocation. For example, existing allocation approaches developed for real-time computing systems and computing clusters [6]–[8] are typically concerned with cyber performance attributes.

To address these challenges faced by shared sensor networks, this paper proposes *Utility-based Multi-application Allocation and Deployment Environment (UMADE)*, an integrated system for application deployment in shared sensor networks. In sharp contrast to traditional approaches that allocate applications based on cyber attributes only, UMADE is based on a cyber-physical system approach that dynamically allocates nodes to applications based on their QoM requirements of the physical environments. Specifically, this work has the following key contributions:

- We propose a novel utility-based approach that allocates nodes to contending applications based on their QoM of the physical environments. A key feature of our approach is the characterization and handling of inter-node QoM dependencies typical in cyber-physical applications.
- We present UMADE, the first integrated system for QoM-aware application deployment in shared sensor networks. UMADE provides an integrated system solution that sup-

ports the end-to-end process of (1) QoM specification for applications, (2) QoM-aware application allocation, (3) application deployment over multi-hop wireless networks, and (4) adaptive reallocation of applications in response to network dynamics.
- We describe the implementation and experimental evaluation of UMADE on a physical testbed of 28 Telos motes in an office building. Our results demonstrate the efficacy and advantages of QoM-aware multi-application deployment in shared sensor networks in the context of building automation applications.

II. Related Works

A. Resource Allocation in Related Domains

Resource allocation has been addressed in several different domains like wireless networks [9], [10], cluster [6], grid [7], and real-time computing systems [8]. The resource allocation problems addressed in these domains are, however, fundamentally different from that addressed by UMADE. For example, wireless networks domain mostly addresses network-level attributes like data rate, packet delay, throughput, and packet-loss-probability, while cluster and grid computing systems address attributes like number of machines and task completion times. These attributes are mostly cyber-oriented and do not address physical aspects such as QoM of physical environments.

B. Utility-based Approaches to Sensor Networks

Utility-based sensor selection schemes presented in [11] and [12] suggest mapping sensor nodesets to utility values. These schemes require the users to directly specify the nodeset to utility mappings and do not provide any QoM abstraction. Moreover, both of these works mostly focus on the theory and algorithms and do not provide any system support for sensor selection. SORA [13] is a utility-based resource allocation system in sensor networks in which nodes act as self-interested agents that select actions to maximize utility under energy constraints. However, SORA is designed for a single application and does not deal with node allocation to multiple applications.

C. System Support for Shared Sensor Networks

With the emergence of shared sensor networks, programming abstractions and systems have been developed to support multiple applications in such systems. Several projects developed group-based abstractions to support multiple concurrent applications [14], [15]. Ma et al. [16] proposed a market-oriented approach for bandwidth allocation in shared sensor networks. However, none of the aforementioned systems for shared sensor networks considers QoM when allocating applications. In contrast, we propose a cyber-physical system approach to application allocation based on the QoM of physical environments.

III. System Model

A. Shared Sensor Networks

A shared sensor network consists of resource-constrained sensor nodes and a base station with more resources. The base station serves as the gateway between the sensor network and the Internet. Users may submit new applications to the shared sensor network through the base station. Applications may be deployed dynamically at different points of time based on user demand. Also, different applications may have different weights based on their importance. For example, the fire detection application may have higher weight than the light monitoring application due to the criticality of the former.

Sensor nodes can be heterogeneous in terms of both supported sensors and resource capacities. A sensor node may be equipped with one or more sensors. Integrating multiple sensors on the same node can reduce hardware cost as the sensors share the microprocessor and radio. A shared sensor network serves as a highly flexible infrastructure that supports different levels of resource sharing among applications. For example, multiple applications may share (1) a sensor on a node (e.g., a magnetometer can be used for detecting parked cars and tracking moving vehicles [2]), (2) a node with multiple sensors, and (3) the network when multiple applications are deployed on different nodes.

Sensor nodes have severe resource constraints. In particular, existing sensor nodes typically have limited memory. For example, the widely used TelosB mote [17] has only 10KB of RAM and 48KB of ROM[1]. As memory is a critical resource in many sensor networks, UMADE is currently designed to deal with memory constraints. In future work we plan to extend UMADE to deal with other resource constraints such as bandwidth and energy.

B. Quality of Monitoring

A shared sensor network employs multiple distributed sensing applications that monitor certain physical phenomena. Many distributed sensing applications are important for cyber-physical systems closely coupled with physical environments. For example, distributed temperature monitoring and estimation are needed for intelligent air conditioning systems, and distributed event detection is needed for security systems.

In contrast to traditional computing applications, the performance of a sensing application should be characterized by its QoM of the physical phenomenon of interest. The QoM attributes are application specific. Here we describe two specific QoM attributes as concrete examples. A contribution of UMADE is that it provides a general framework for characterizing and incorporating a wide range of QoM attributes suitable for different applications.

- *Variance reduction:* Many distributed sensing applications are designed to estimate *spatially correlated* phenomena (e.g., temperature and humidity). For exam-

[1]While some sensor nodes have external flash with a capacity of up to a few megabytes, writing to the flash is power consuming and cannot be used in place of memory for many applications.

ple, intelligent air conditioning requires fine-grained estimation of the temperature distribution based on the measurement of a finite set of nodes. The temperature measurements of different sensors are correlated with each other, and the degree of their correlations depends on the sensor locations and the spatial distribution of temperature. To exploit the correlation of sensor readings, probabilistic models have been developed that enable the prediction of sensor readings at all nodes based on the sensor readings of a subset of nodes. Such probabilistic methods are fairly general and have been validated in a broad range of applications that monitor temperature, humidity, and pollution in waste water [5], [18]–[20]. While they have been used for sensor selection for target localization [21] and for sensor placement [5], [18]–[20], we propose to use them, for the first time, for allocating subsets of nodes to multiple contending applications. Different subsets of nodes provide different reductions in the variance of the estimated sensor readings. The higher the variance reduction, the higher the confidence in the predictions. Thus, *variance reduction* is an important QoM attribute for a common class of distributed sensing applications that aim to estimate the spatial distributions of environmental variables by sampling a subset of nodes.

- *Detection probability:* Detection probability is a common QoM attribute for event detection applications. It is defined as the probability for a specific type of event to be detected by a fusion group consisting of multiple sensors, subject to a certain upper bound of false alarm rate. For instance, with distributed detection based on decision fusion, the local detection probability of a node can be estimated based on a stochastic model of the signals and noise. The per-node detection probabilities are then combined to obtain the system detection probability based on the fusion rule. Different subsets of nodes provide different detection probabilities, depending on the locations of the nodes in the subset.

As discussed in the above examples, an inherent property of distributed sensing applications is that the measurements of a physical phenomenon from different sensors are usually correlated to each other. As a result, the contribution of a node to the QoM of an application is dependent on the other nodes allocated to the same application. We call this property *inter-node QoM dependency*. For instance, the contribution of a sensor to the variance reduction for temperature estimation application is heavily influenced by the correlation of its measurement with those of sensors allocated to the same application. Similarly, the contribution of a sensor to system detection probability depends on the locations of sensors allocated to the same fusion group [22]. Handling such inter-node dependencies of QoM is a key contribution and novelty of the UMADE system which distinguishes it from traditional application allocation approaches.

Note that, besides the QoM attributes, UMADE also supports traditional computing attributes (e.g., reliability) that are sometimes needed for some applications. UMADE computes node reliability as the product of the sensor and network path reliability. The reliability achieved by an application is the probability that at least one node among the set of nodes assigned to it is alive and successfully sending data to the base station.

C. Utility Function

To handle dynamic application arrivals, a shared sensor network should support flexible tradeoff between QoM and the resource consumption of an application. UMADE allows users to specify an application's utility at different QoM levels through its utility function [8], [10], [16]. A *utility function* specifies the range of QoM acceptable by the application and the corresponding utility gained by the application. Therefore, this function represents a QoM to utility mapping. Utility functions are non-decreasing and typically concave since sensing applications tend to have diminishing marginal returns on QoM. While these functions can be arbitrarily complex, simple functions (e.g., piece-wise linear) usually suffice for sensing applications. We require utility functions to be only non-decreasing. For example, a simple utility function for an application that requires a minimum variance reduction of 80% and a maximum variance reduction of 95% can be represented as shown in Fig. 1.

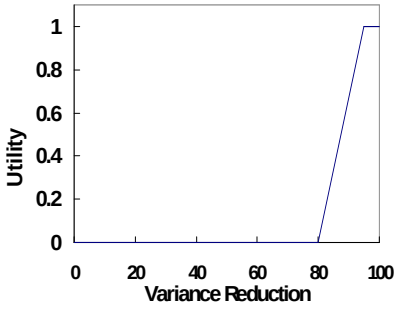


Fig. 1. Example utility function

D. Allocation Objective

The objective of application allocation by UMADE in a shared sensor network is to allocate sensor nodes to applications so as to maximize the system utility subject to node memory constraints. This problem can be formulated as follows:

1) The sensor network consists of m nodes, denoted by the set $R = \{R_1, R_2, \cdots, R_m\}$. Let L_k be the memory available at node R_k, $1 \leq k \leq m$.
2) A total of n applications, denoted by the set $A = \{A_1, A_2, \cdots, A_n\}$, need to be deployed. Each application A_j, $1 \leq j \leq n$, has a weight w_j, memory requirement r_j, and an associated utility function $u_j : Q_j \rightarrow U$, where Q_j is the set of possible *QoM* values that can be received by A_j, and $U = \{x | 0 \leq x \leq 1\}$ is the set of *utility* values.
3) For every application A_j, $1 \leq j \leq n$, there is a QoM profile that maps a *nodeset* to a *QoM* value and is

represented by $q_j : \mathcal{P}(R) \rightarrow \mathbb{R}$. That is, if application A_j is deployed on a nodeset $S_j \subseteq R$, then it achieves a QoM of $q_j(S_j)$ and a utility of $u_j(q_j(S_j))$.

The *total system utility* is defined as the weighted sum of the utilities of n applications, i.e., $\sum_{j=1}^{n} w_j * u_j(q_j(S_j))$. Thus, the objective of the QoM-aware allocation is to assign a nodeset $S_j \subseteq R$ to each application A_j so as to

$$maximize \sum_{j=1}^{n} w_j * u_j(q_j(S_j)) \quad (1)$$

$$subject\ to \sum_{j=1}^{n} a_{jk} * r_j \leq L_k,\ \forall k = 1, \cdots, m;$$

where a_{jk} is 1 if node R_k is allocated to application A_j, and 0 otherwise. This constraint states that the total memory consumed by the applications sharing a node must not exceed the memory available at the node.

This allocation problem for shared sensor networks is NP-hard, which can be proven through a straightforward reduction from the QoS optimization problem stated in [8] which has been shown to be NP-hard. Due to space limit, the proof for NP-hardness is not included in this paper and can be found in a technical report [23].

It is important to note that this QoM-aware application allocation for shared sensor networks is *fundamentally different* from traditional QoS-based resource allocation (Q-RAM) [8] and multiple knapsack problems (MKP) [24], because shared sensor networks must deal with *inter-node QoM dependencies* imposed by distributed sensing applications. In contrast, neither Q-RAM nor MKP considers inter-node dependencies in resource allocation. In MKP, the items in one knapsack do not affect the weight of items in adjacent knapsacks. Similarly, the QoS values achieved by different nodes are considered independent from each other in Q-RAM. The key novelty and contribution of our work is to model and handle such inter-node dependencies imposed by distributed sensing applications. For example, since inter-node QoM dependency implies that the QoM value of a set of nodes is not equivalent to the summation of the QoM values of the individual nodes, the above problem formulation uses $nodeset$ to QoM mappings to capture the inter-node QoM dependencies in distributed sensing applications. Furthermore, as described in the next section, the UMADE system supports automatic mapping of nodeset to QoM based on training data, and employs efficient greedy heuristics to allocate applications while considering the inter-node QoM dependencies.

IV. System Design and Implementation

The UMADE system, as shown in Fig. 2, mainly consists of the *Application Allocation Engine* and the *Application Deployment Engine*. The allocation engine completely resides at the base station (since applications are usually submitted at a base station) while the deployment engine is distributed across the base station and sensor nodes.

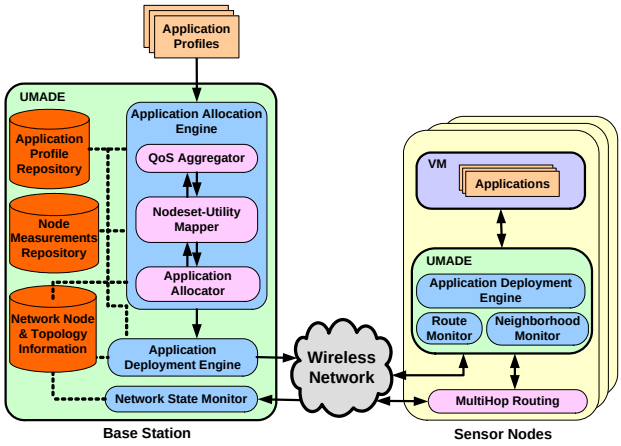


Fig. 2. UMADE architecture

The Application Allocation Engine is invoked whenever applications are submitted to the system for deployment. Applications are submitted using application profiles. An application profile includes the application *code*, *weight*, *memory requirement*, and *utility function*. The application memory requirement including memory required for the application code, data, and other system variables can be obtained through simulation and analysis tools [25], [26]. After application submission, the allocation engine executes an allocation algorithm (described in Subsection IV-B) to allocate sensor nodes to the applications. The output of the Application Allocation Engine specifies a set of nodes for each application A_j, $1 \leq j \leq n$, meaning that application A_j should be deployed on this set of nodes. This output is then sent to the deployment engine that deploys the applications wirelessly in the shared sensor network according to the allocation. Following are the details of the design and implementation of the key components of UMADE.

A. Nodeset to QoM Mapping

UMADE automatically generates $nodeset$ to QoM mappings, thereby easing the task of the users. It achieves this by supporting pluggable modules that compute the $nodeset$ to QoM mappings based on training data consisting of measurement from individual sensor nodes. These pluggable modules act as *QoM Aggregators* that aggregate and map individual sensor measurement to system QoM attributes. For example, a QoM Aggregator for variance reduction outputs the variance reduction achieved by a nodeset based on the Gaussian joint distribution estimated using the training data from individual sensor nodes. A QoM Aggregator for detection probability outputs the system detection probability achieved by a nodeset based on the fusion rule and the training data (the local detection probabilities) of individual sensor nodes. The training data may be provided by the user based on historical data, or automatically collected by a sensor data collection tool provided by UMADE. Note that a complete $nodeset$ to QoM mapping has an exponential size of 2^m, where m is the total number of nodes. Hence, UMADE does not precompute the entire mapping for each application. Instead, it generates

only the required mappings on the fly, when required by the allocation algorithm, that helps in keeping the overall time and space polynomial.

For example, the QoM Aggregator for variance reduction uses the following procedure to map a nodeset, S, to the variance reduction that it provides.

1) It collects training data from all sensor nodes in the network. This data consists of readings from all sensors that are used for sensing spatial phenomena like light, temperature, and humidity.
2) For each sensor type, it assumes that the corresponding sensor values have a (multivariate) Gaussian joint distribution and computes a covariance matrix Σ_V, where V is the set of all nodes that have the corresponding sensor type [5]. For example, a unique Σ_V is computed for each of the temperature, light, and humidity sensor types.
3) It uses the covariance matrix to compute the variance reduction $q(S) = trace(\Sigma_V) - trace(\Sigma_{V \setminus S|S})$ achieved by subset $S \subseteq V$. The exact equations used for computing Σ_V and $\Sigma_{V \setminus S|S}$ can be found in [5].

This procedure is based on the probabilistic method using Gaussian processes proposed in [5] which was originally designed for optimal sensor placement. Due to the generality of the probabilistic method, this QoM Aggregator can be applied to a wide range of environmental variables, such as temperature, humidity, and water pollution. UMADE also allows users to plug in other QoM Aggregators to support other QoM attributes.

B. QoM-aware Application Allocation

The allocation engine invokes the QoM-aware Application Allocation Algorithm whenever a set of new applications is submitted. Since the optimal allocation problem is NP-hard as discussed in Section III, we propose a simple greedy heuristic to assign a nodeset to each application. Note that the allocation algorithm needs to be reasonably efficient because the allocation needs to be recomputed dynamically in response to application arrivals and departures, as well as node additions and removals. As noted in Section III, existing heuristics for Q-RAM [8] and MKP [24] are not suitable for shared sensor networks as they do not consider inter-node QoM dependencies. The novelty of our QoM-aware Application Allocation Algorithm lies in its capability to handle inter-node QoM dependencies. Moreover, it is integrated with the QoM Aggregator that provides automatic nodeset to QoM mapping needed to capture the inter-node QoM dependencies.

The greedy allocation algorithm works as follows. First, a list of *available nodes* that have enough free memory to accommodate at least one of the applications is obtained. Next, nodes from this list are repeatedly assigned to the *pending applications* until all applications achieve their maximum desired utility values or until there are no more available nodes. An application is called a *pending application* if it has not yet achieved its maximum utility specified by its utility function. Nodes are assigned as follows: each available node is considered in turn and assigned to the pending application that (1) has not been assigned that node before, (2) has a memory requirement that can be satisfied by the node, and (3) when assigned, results in the maximum increase in system utility per unit memory consumption. The increase in system utility upon allocation of a node R_i to an application A_j (needed in step 3) is obtained by multiplying the weight of application A_j to the increase in utility, say Δu_j, of application A_j. Let S be the subset of nodes containing the new node R_i and the other nodes that have already been allocated to A_j. Then, $\Delta u_j = u_j(q_j(S)) - u_j(q_j(S - \{R_i\}))$, where $u_j(q_j(S))$ is the utility provided to A_j by nodeset S and $u_j(q_j(S - \{R_i\}))$ is the current utility of A_j (see Subsection III-D). The pseudo code of our allocation algorithm is shown in Algorithm 1.

As shown Fig. 2, to get the utility achieved by a nodeset the Application Allocator invokes the Nodeset-Utility Mapper, which in turn invokes the QoM Aggregator to get the QoM value provided by the nodeset and then maps the QoM value to the utility value based on the utility function specified for the application. Note that our allocation algorithm calculates the utility gain based on the *nodeset* to *QoM* mapping, which enables it to capture the inter-node QoM dependencies imposed by distributed sensing applications.

```
A' ← set of applications;
R' ← set of nodes that can accommodate an application;
while A' ≠ ∅ and R' ≠ ∅ do
    for each R_i ∈ R' do
        max = 0;
        for each A_j ∈ A' which can be accommodated by R_i
        and has not yet been assigned R_i do
            S ← set of R_i and nodes assigned to A_j;
            Δu_j = u_j(q_j(S)) − u_j(q_j(S − {R_i}));
            if w_j * Δu_j > max then
                max = w_j * Δu_j; A_max = A_j;
            end
        end
        Assign node R_i to application A_max;
        if A_max achieves its maximum utility then
            A' = A' − {A_max};
        end
    end
    for each R_i ∈ R' do
        A'' ← all applications in A' which have not been
        assigned to R_i;
        if R_i does not have enough memory to accommodate
        any A_j ∈ A'' then R' ← R' − {R_i};
    end
end
```

Algorithm 1: QoM-aware Application Allocation

Since the greedy algorithm is affected by the ordering of the available nodes, we execute the algorithm for a fixed number of rounds, say I, while varying the ordering of the available nodes in each round. The best solution over all rounds is selected as the final solution. The greedy solution, thus, has a polynomial time complexity of $O(n^2 m)$, where m is the number of nodes and n is the number of applications and, therefore, scales to large number of nodes and applications

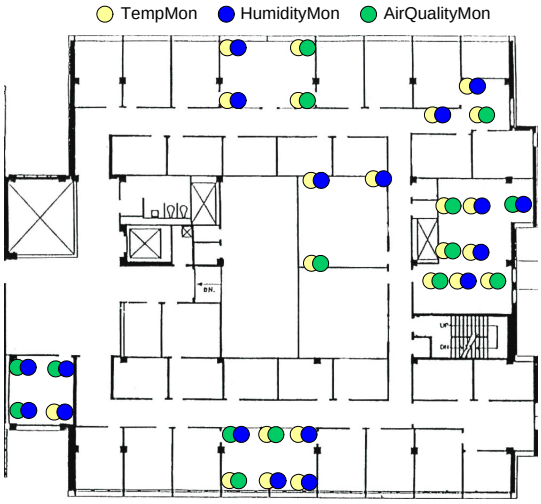


Fig. 3. Application allocation (deployment 1 on our testbed)

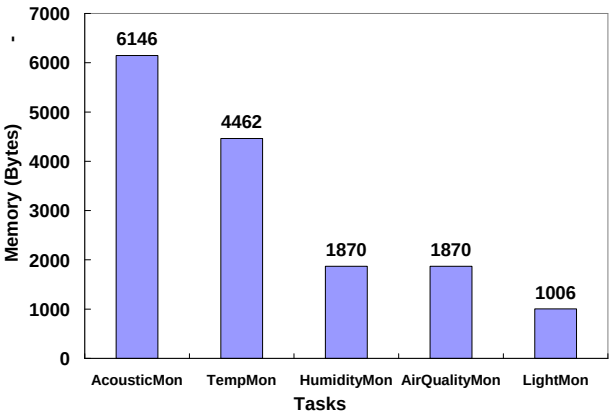


Fig. 4. Application memory requirements

(as shown in our simulation results in Section VI). Due to the non-linearity and discrete nature of the allocation problem as a result of the inter-node QoM dependencies, we have not proven the approximation bound of our greedy algorithm. Our evaluation (Section V) indicates that it achieves system utilities that are only slightly worse than that achieved by the optimal algorithm under realistic settings.

C. Handling Application and Network Dynamics

UMADE adapts application allocations in response to (1) dynamic application arrivals and terminations and (2) sensor node additions and removals/failures. To balance system utility and redeployment cost, UMADE considers both *preemptive* and *non-preemptive* allocation strategies when computing the new allocation. Preemptive allocation recomputes the allocations of all applications across the entire network and, hence, may reallocate existing applications. In contrast, non-preemptive allocation only considers new applications and nodes with sufficient residual memory when computing the new allocation and, hence, does not reallocate applications already deployed on existing nodes. If the system utility resulting from preemptive allocation exceeds that of the non-preemptive allocation by more than a threshold, then UMADE deploys applications according to the preemptive allocation. Otherwise, UMADE picks the non-preemptive allocation. To enable the user to adjust the tradeoff between utility and deployment cost, UMADE can generate different allocations with a range of thresholds (as shown in simulation results in Section VI) for system administrators to choose from. Note that this threshold-based approach is practical since the allocation is computed at the base station and the allocation algorithm has polynomial complexity.

D. Implementation

UMADE has been implemented as an integrated environment that supports the end-to-end process of application deployment in shared sensor networks. The Allocation Engine on the base station is implemented in Java, except the QoM Aggregator which is implemented in MATLAB. The Application Deployment Engine on the sensor nodes is written in NesC [27] on the TinyOS [28] operating system. It currently employs the *Agilla VM* [29] to support concurrent application execution and dynamic application deployment, although it may be extended to work with operating systems that support dynamically loadable modules [30], [31]. The Agilla VM supports applications implemented in high-level scripts, thereby enabling low cost re-tasking of the sensor network. We extended the Agilla VM to support dynamic memory management in order to support applications with a range of memory requirements. As a result, we were able to reserve 7KB out of 10KB of RAM available on a TelosB mote for the applications. Multi-hop routing is achieved using the *MultiHopLQI* protocol [32]. MultiHopLQI builds a routing tree rooted at the base station in which parent nodes are selected based on the link LQI values. Information about the routing tree is collected periodically at the base-station. This information is used to compute routes from the base station to the nodes, which are used for source-routing-based deployment of applications to designated nodes over multi-hop network. On a TelosB mote, the UMADE system uses 2626B of RAM for itself, while making 7KB available to applications running on Agilla VM. It uses 45494B of ROM.

V. Testbed Experiments

In this section, we present an empirical evaluation of the entire UMADE system on a physical testbed consisting of 28 TelosB motes and one Pentium IV PC acting as the base station. These experiments evaluate UMADE using realistic applications in the context of building automation system. In the next section we evaluate the scalability and the threshold-based dynamic deployment strategy under a broader range of conditions through simulations.

Our 28-node testbed covers 6 rooms on the fifth floor of Jolley Hall in Washington University and forms a 3-hop wireless sensor network. Fig. 3 shows the layout of our testbed. We evaluated UMADE using five representative applications in the context of building automation: temperature monitoring (TempMon), humidity monitoring (HumidityMon), air quality monitoring (AirQualityMon), light monitoring for lighting

control (LightMon), and acoustic signal monitoring for noise control (AcousticMon). The applications periodically sample and transmit the average sensor data to the base station. The memory requirements of the applications are shown in Fig. 4.

We implemented the temperature, light, and humidity monitoring applications. Due to the lack of required sensors on our testbed, we emulate the other two applications based on estimated memory requirement. The memory requirement of the applications is dependent on the sensor data size. The sensor data size is set to 16 bits assuming that the sensors are connected to a 12-bit ADC based on the information provided in the data sheets of commercially available acoustic and air quality sensors. The weights of all five applications were initially set to 1. *Variance reduction* was used as the QoM attribute for the first three applications, since all three applications monitor spatial phenomena. In order to evaluate the flexibility of our system in handling applications with different attributes, we use *reliability* metric (as defined in Section III) for the other two applications.

Fig. 1 shows the utility function used for the TempMon application. The utility functions used for other applications can be found in [23]. In practice, the utility functions should be assigned based on the application characteristics and user requirements. For example, given the importance of TempMon in achieving a high comfort level through intelligent air conditioning, TempMon may be assigned a high variance-reduction (QoM) requirement ranging from 80% to 95% as shown in Fig. 1. We computed the $nodeset$ to QoM mappings for the applications in the following way. For the first three applications, we collected the temperature, light, and humidity readings from all testbed nodes over a few hours in order to compute the temperature, light, and humidity covariance matrices, which are then used to compute the nodeset to variance reduction mappings for TempMon, LightMon, and HumidityMon, respectively. To obtain the nodeset to reliability mappings for AirQualityMon and AcousticMon, we calculated the per-node failure probability and computed the reliability provided by a set of nodes as $1 - (product\ of\ failure\ probability\ of\ all\ nodes\ in\ the\ set)$. The failure probability of a node was computed as $1 - (node\ reliability)$. Node reliability, was calculated based on the reliability of the sensor used (assumed to be 0.98 in our experiments), and the reliability of the multi-hop path from the node to the base station which was computed using per-link LQI readings.

We have evaluated the performance of our greedy algorithm against an optimal algorithm that uses exhaustive enumeration to obtain an optimal allocation that maximizes system utility under the memory constraints. We have also compared the performance of our greedy algorithm against two standard application allocation algorithms - *knapsack* and *randomized* allocation. The knapsack algorithm sorts the applications (in non-increasing order) according to the ratio of their weights and their memory requirements and uses this list to sequentially assign applications to each sensor node. Thus, the knapsack algorithm achieves a uniform allocation on all the sensor nodes. The randomized-allocation algorithm, on the other hand, randomly picks applications to assign to each network node. The allocations are computed per room since this physical division enables the use of a divide and conquer strategy of the algorithms.

We have evaluated the performance of UMADE using a step-wise deployment scheme. Each deployment was initiated after an arbitrary interval of 1 hour. In the first step (Deployment 1), we deployed three applications - TempMon, HumidityMon, and AirQualityMon. In the second step (Deployment 2), we increased the number of applications by adding LightMon and AcousticMon. In the third step (Deployment 3), we changed the weight of one of the applications to evaluate its effect on the application allocations. The results obtained in each step are presented below.

A. Deployment 1: Comparison of Allocation Methods

Fig. 3 shows the allocation of the three applications in the testbed after they are deployed by UMADE based on our QoM-aware greedy algorithm. Given the memory requirements of TempMon, HumidityMon, and AirQualityMon applications, it is easy to see that at most two of these applications fit on a node (since only 7K is available per node for applications). Fig. 5(a) compares the system utility achieved by our greedy algorithm with the optimal, knapsack, and randomized allocation algorithms. Fig. 5(a) shows that in some cases the greedy algorithm achieves the optimal system utility while in other cases it performs only slightly worse than the optimal algorithm. The greedy algorithm, however, consistently performs much better than the knapsack and randomized-allocation algorithms, which is expected, since these two algorithms do not try to optimize the system utility. Fig. 5(b) compares the execution times of these algorithms. The greedy algorithm is several orders of magnitude faster than the optimal algorithm. For example, for Room 2 which has 8 sensor nodes, the optimal algorithm takes 3205 seconds as compared to 2.7 seconds taken by the greedy algorithm. The greedy algorithm is slower than the knapsack and the randomized-allocation algorithms since it executes for a number of iterations.

Given that the greedy algorithm achieves a much higher system utility than both the knapsack and randomized-allocation

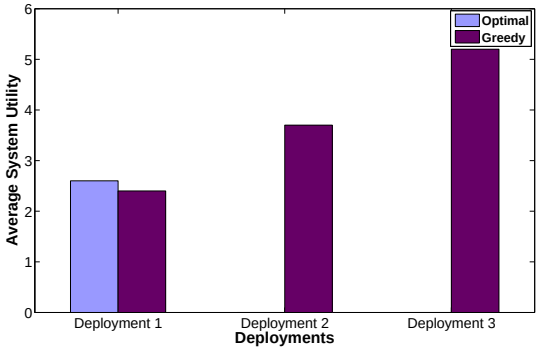
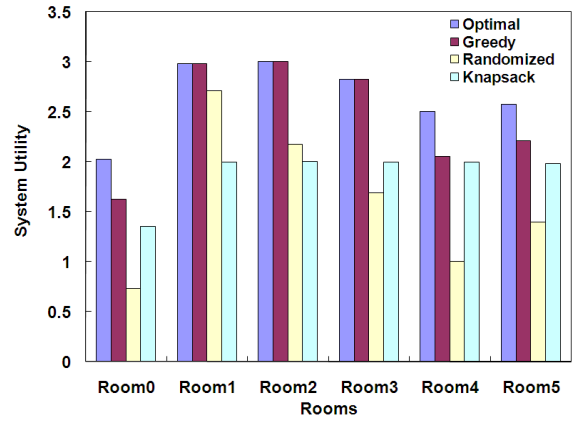
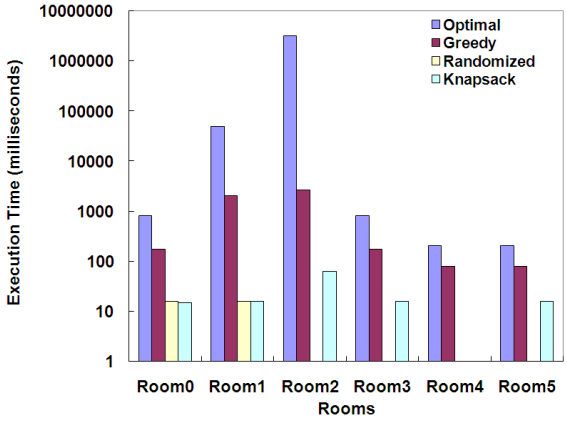


Fig. 6. Average system utility per room

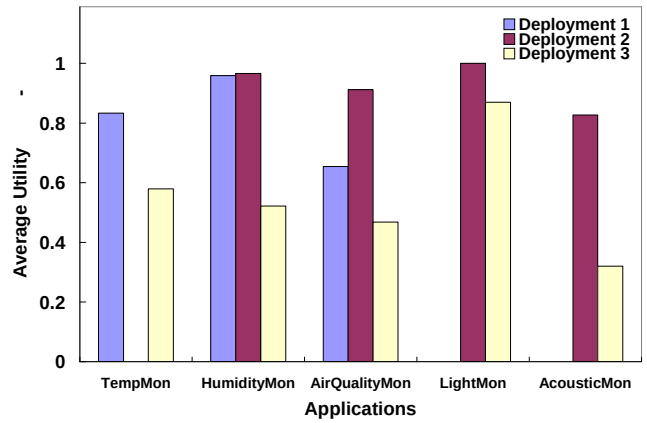


(a) System utility

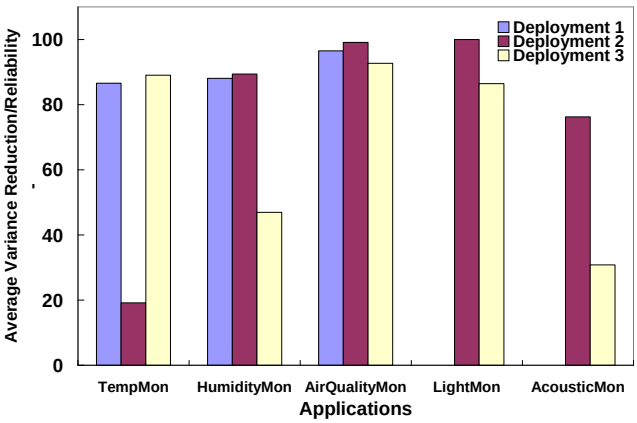


(b) Execution time (note that missing bars indicate 0 values)

Fig. 5. Performance comparison of algorithms during deployment 1.



(a) Application utility



(b) Application variance reduction/reliability

Fig. 7. Application utility and variance reduction/reliability achieved per deployment

algorithms, we exclude these two schemes in later experiments. Fig. 6 shows the average system utility per room achieved by the greedy and optimal algorithms.

B. Deployment 2 and 3: Adaptation to Changes

Deployment 2 tests how UMADE adapts when two new applications are added to the system. Fig. 6 shows that the overall system utility is improved as a result of the new allocations computed by UMADE. The figure only shows the greedy solution because the optimal algorithm has not terminated after several hours of execution. The individual application utilities and variance reduction/reliability values achieved with the new allocation are shown in Fig. 7(a) and Fig. 7(b), respectively. Note the change in the application utilities. In particular, we see that the TempMon application is not allocated enough nodes resulting in low QoM and 0 utility. This is because of the relatively high QoM and memory requirements of the application.

In Deployment 3, we increase the importance of the TempMon application by increasing its weight to 5 while maintaining the weights of the other applications at 1. The improvement in the utility and QoM achieved by the TempMon application as a result of this change is shown in Fig. 7(a) and Fig. 7(b), respectively. Notice how the TempMon utility increases to about 60% from 0 and the QoM increases to 90% from 19%. The improvement in the overall system utility achieved as a result of this change is shown in Fig. 6. These experiments demonstrate that the UMADE system can effectively adapt to dynamic changes to the applications.

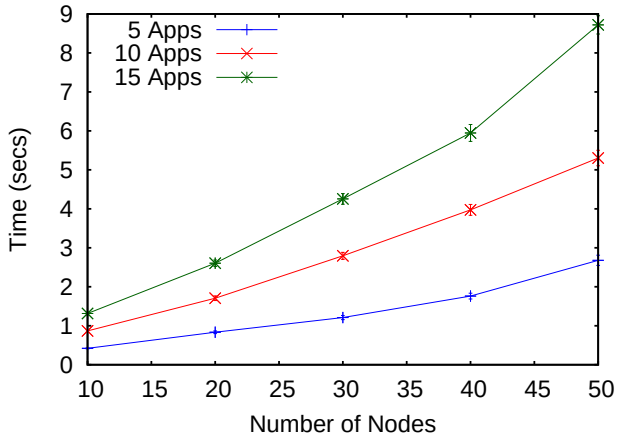
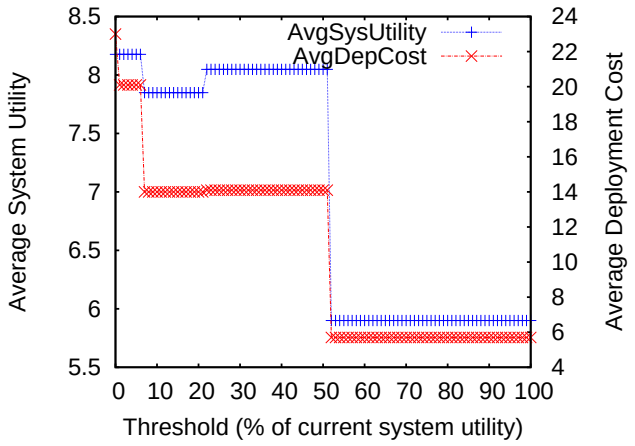
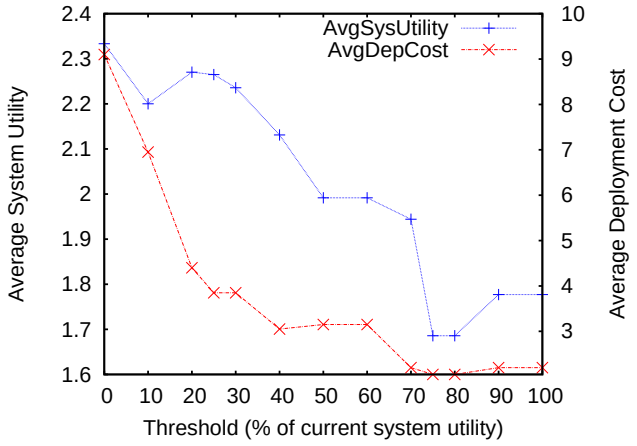


Fig. 8. Execution time under varying network size and number of applications



(a) Performance under increasing number of applications



(b) Performance under increasing network size

Fig. 9. Performance under varying network size and number of applications

VI. Simulations

In this section, we present an evaluation of UMADE with a wide range of synthetic workloads and settings. Because of the extremely long run-time needed by the optimal algorithm, we do not compare the QoM-aware greedy allocation algorithm against the optimal algorithm in simulations because they involve large numbers of nodes and applications. In the simulation setup, each application is assigned a random weight between 1 to 6, a random memory requirement ranging from 1000 bytes to 7 KB, and a random two-segment utility function. *Nodeset* to *QoM* mappings are also generated at random. Each node has a maximum memory capacity of 7KB for applications.

The first set of simulations evaluates the scalability of UMADE's QoM-aware greedy allocation algorithm with different number of nodes and applications. Fig. 8 illustrates how the execution time of the greedy allocation algorithm on a Pentium IV PC varies with the numbers of nodes and applications. Each data point in the figure is averaged over 100 runs. 95% confidence intervals are also plotted but are not visible due to the small range of the intervals. The figure indicates that the greedy algorithm can scale effectively to large cluster sizes and number of applications.

The second set of simulations evaluates the cost-effectiveness of the threshold-based allocation scheme in the face of application and node additions and removals. Fig. 9(a) shows UMADE's performance when the number of applications is increased from 1 to 10 on a 30-node network. We deploy one application at a time and compute the *deployment cost* as the summation of the number of new nodes assigned to each application. We also compute the average system utility and average deployment cost over different configurations of the network. The figure shows that both the average system utility and average deployment cost reduce as the threshold is varied from 0 to 100% utility-gain which is expected. Interestingly, when the threshold is between 25% and 50%, the drop in deployment cost is much larger (38.7%) than the drop in system utility (1.6%). Hence, a threshold-based allocation scheme would result in only 1.6% reduction in system utility when compared against the preemptive scheme, while reducing the deployment cost by 38.7%. This result demonstrates that, by exploring different threshold values for each deployment, UMADE enables the user to identify the most cost-effective allocation scheme when a new application arrives.

Fig. 9(b) shows UMADE's performance when the number of applications is fixed at 10, but the number of nodes is increased from 10 to 30. When a new node is added to the system, UMADE uses a policy similar to what it does when a new application is submitted to the system. That is, it redeploys all applications if the system utility can be increased to a value greater than a threshold; otherwise it just deploys applications on the new node if the utility gain is less than the threshold (but > 0). Here, too, we see that the average system utility and deployment cost drop as the threshold is increased from 0 to 100% utility-gain, as expected. In this case, thresholds in the range 20% to 25% give the best balance between utility gain and deployment cost.

Thus, we see that the threshold is an effective knob for achieving a desired balance between utility gain and deployment cost. Exposing the system utility and deployment cost with different thresholds to the user allows the user to choose a threshold (per deployment) that results in the desired tradeoff between deployment cost and system utility.

VII. Conclusion

As wireless sensor networks evolve into integrated infrastructure shared by numerous applications, they face the critical need for allocating and deploying applications in shared networks. We have developed UMADE, an integrated environment for application deployment in shared sensor networks. In sharp contrast to traditional approaches that allocate applications based on computing metrics only, UMADE adopts a cyber-physical system approach to allocate applications based on the *Quality of Monitoring (QoM)* of the physical environments. The key novelty of UMADE is that it is designed to deal with the *inter-node QoM dependencies* inherent to many distributed sensing applications. Furthermore, UMADE provides an *integrated system solution* that supports

(1) automatic nodeset to QoM mapping for applications, (2) QoM-aware application allocation, (3) application deployment over multi-hop wireless networks, and (4) adaptive reallocation of applications. UMADE has been implemented on TinyOS and the Agilla virtual machine for Telos motes. The efficacy of UMADE has been demonstrated on a 28-node wireless sensor network testbed in the context of building automation applications.

As a promising start towards supporting integrated sensing systems, UMADE also opens up several research directions. UMADE currently treats memory as the critical resource constraint for shared sensor networks. In the future, UMADE will be extended to address multiple resource constraints. In addition, UMADE currently adopts a centralized approach to application allocation, which can effectively handle our 28-node network in our empirical study. To handle large-scale networks we will investigate hierarchical approaches. For example, in a building automation system for a large commercial building, different floors may form separate sub-networks connected by an upper-tier network. UMADE can be extended to perform multi-level application allocation in a hierarchical fashion in such tiered networks.

ACKNOWLEDGEMENT

This research was supported by NSF under grants CNS-0520220 (NeTS-NOSS), CNS-0627126 (NeTS-NOSS), and CNS-0708460 (CRI).

REFERENCES

[1] R. Murty, G. Mainland, I. Rose, A. Chowdhury, A. Gosain, J. Bers, and M. Welsh, "CitySense: An urban-scale wireless sensor network and testbed," *IEEE Conference on Technologies for Homeland Security*, pp. 583–588, May 2008.

[2] "Can't find a parking spot? Check smartphone," July 2008, http://www.nytimes.com/2008/07/12/business/12newpark.html.

[3] "Building automation applications," 2006, Ember Corporations, http://www.ember.com/applications_building_automation.html.

[4] M. A. Batalin, M. Rahimi, Y. Yu, D. Liu, A. Kansal, G. S. Sukhatme, W. J. Kaiser, M. Hansen, G. J. Pottie, M. Srivastava, and D. Estrin, "Call and response: experiments in sampling the environment," in *SenSys*, 2004.

[5] C. Guestrin, A. Krause, and A. P. Singh, "Near-optimal sensor placements in gaussian processes," in *ICML*, 2005.

[6] "PlanetLab: An open platform for developing, deploying, and accessing planetary-scale services," http://www.planet-lab.org.

[7] D. Abramson, J. Giddy, and L. Kotler, "High performance parametric modeling with Nimrod/G: Killer application for the global grid?" in *IPDPS*, 2000.

[8] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, and J. Hansen, "A scalable solution to the multi-resource QoS problem," in *RTSS*, 1999.

[9] T. B. Reddy, I. Karthigeyan, B. S. Manoj, and C. S. R. Murthy, "Quality of service provisioning in ad hoc wireless networks: a survey of issues and solutions," *Ad Hoc Networks*, vol. 4, no. 1, pp. 83–124, 2006.

[10] C. Curescu and S. Nadjm-Tehrani, "Price/utility-based optimized resource allocation in wireless ad hoc networks," in *SECON*, 2005.

[11] F. Bian, D. Kempe, and R. Govindan, "Utility based sensor selection," in *IPSN*, 2006.

[12] J. Byers and G. Nasser, "Utility-based decision-making in wireless sensor networks," in *MobiHoc*, 2000.

[13] G. Mainland, D. C. Parkes, and M. Welsh, "Decentralized, adaptive resource allocation for sensor networks," in *NSDI*, 2005.

[14] C. Frank and K. Römer, "Algorithms for generic role assignment in wireless sensor networks," in *SenSys*, 2005.

[15] Y. Yu, L. J. Rittle, V. Bhandari, and J. B. LeBrun, "Supporting concurrent applications in wireless sensor networks," in *SenSys*, 2006.

[16] Q. Ma, D. C. Parkes, and M. Welsh, "A utility-based approach to bandwidth allocation and link scheduling in wireless networks," in *ATSN*, 2007.

[17] "TelosB," http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/TelosB_Datasheet.pdf.

[18] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong, "Model-driven data acquisition in sensor networks," in *VLDB*, 2004.

[19] A. Krause, J. Leskovec, C. Guestrin, J. VanBriesen, and C. Faloutsos, "Efficient sensor placement optimization for securing large water distribution networks," *Journal of Water Resources Planning and Management*, vol. 134, no. 6, pp. 516–526, November 2008.

[20] A. Krause, B. McMahan, C. Guestrin, and A. Gupta, "Robust submodular observation selection," *Journal of Machine Learning Research (JMLR)*, vol. 9, pp. 2761–2801, December 2008.

[21] V. Isler and R. Bajcsy, "The sensor selection problem for bounded uncertainty sensing models," in *IPSN*, 2005.

[22] G. Xing, C. Lu, R. Pless, and J. A. O'Sullivan, "Co-Grid: an efficient coverage maintenance protocol for distributed sensor networks," in *IPSN*, 2004.

[23] S. Bhattacharya, A. Saifullah, C. Lu, and G.-C. Roman, "Multi-application deployment in integrated sensing systems based on quality of monitoring," Washington University in St. Louis, Tech. Rep. WUCSE-2008-17, 2008.

[24] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. New York, John Wiley and Sons Ltd., 1990. [Online]. Available: http://www.or.deis.unibo.it/knapsack.html

[25] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and scalable simulation of entire TinyOS applications," in *SenSys*, 2003.

[26] J. Regehr, A. Reid, and K. Webb, "Eliminating stack overflow by abstract interpretation," *ACM Transactions in Embedded Computing Systems (TECS)*, vol. 4, no. 4, pp. 751–778, 2005.

[27] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesC language: A holistic approach to networked embedded systems," in *PLDI*, 2003.

[28] "TinyOS Community Forum," http://www.tinyos.net/.

[29] C.-L. Fok, G.-C. Roman, and C. Lu, "Rapid development and flexible deployment of adaptive wireless sensor network applications," in *ICDCS*, 2005.

[30] C.-C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava, "A dynamic operating system for sensor nodes," in *MobiSys*, 2005.

[31] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Emnets*, 2004.

[32] "MultiHopLQI," 2004, http://www. tinyos.net/ tinyos-1.x/ tos/ lib/ MultiHopLQI/.