Supporting Multi-Fidelity-Aware Concurrent Applications in Dynamic Sensor Networks

Nirmalya Roy, The University of Texas at Austin
Vasanth Rajamani, The University of Texas at Austin
Christine Julien, The University of Texas at Austin

TR-UTEDGE-2010-010

Supporting Multi-Fidelity-Aware Concurrent Applications in Dynamic Sensor Networks

Nirmalya Roy, Vasanth Rajamani and Christine Julien
The University of Texas at Austin
Email: {nirmalya.roy, vasanthrajamani, c.julien}@mail.utexas.edu

Abstract— **Most existing research in wireless sensor networks focuses on optimally running a single application on top of a tailor-made and deployed network. However, as sensors become an integral part of our environments, we posit that sensor networks will be increasingly viewed as platforms that will be used to run several user applications simultaneously. Concurrently executing applications requires the network's resources to be shared across applications so as to make the best long-term utilization of the constrained devices and communications network. This resource sharing entails tradeoffs in the fidelity offered to individual applications. Fidelity is an application-dependent concept that can denote a variety of operational measures including communication latency, data quality, and redundancy. In this paper, we present a principled way to define the fidelity associated with an application given a particular allocation of the available resources. Given a desired set of applications to deploy, we also explore the impact of sharing resources among the applications in terms of fidelity degradation to individual applications. To this end, we provide an algorithm that determines the optimal subset of applications to deploy given the available resources and the potential impact on fidelity and evaluate it.**

I. Introduction

Enabling multiple applications to execute concurrently in sensor networks requires allocating the network's resources to the applications' potentially competing tasks in a way that maximizes the applications' fidelities, or qualities of performance. As sensor networks become ubiquitous, they will increasingly be treated as shared platforms for application-dependent sensing, computation, storage, and communication. This diverges from the current paradigm of designing individual applications optimized for a particular sensor network deployment. However, sharing a common pool of resources between different applications may result in individual applications performing suboptimally, if at all. In this paper, we present the theoretical underpinnings of a sensor network architecture that can navigate the tensions between resource sharing and the potential ensuing fidelity degradations. To gain a deeper understanding of the tradeoff, we first identify the following interrelated issues:

Resource Sharing. The basic premise behind any resource sharing scheme is to identify common tasks across different applications. Output from these shared tasks can be used across multiple applications instead of each application executing the task independently. This requires a fundamental understanding of the nature and scope of a *task*.

Tasks. Tasks are units of execution that make up an application. We abstract the tasks that occur in a sensor network into three classes: *capture*, *storage*, and *distribution*. Capture refers to the process of generating data, both in terms of raw data sensing and data aggregation (both temporal and spatial). Storage refers to the maintenance of captured values for finite periods of time. Finally, distribution refers to tasks that utilize the communication medium to move captured values among the devices. A single application may perform a variety of combinations of these tasks. To understand how the network's resources can be shared to accomplish applications' tasks requires understanding the quality with which applications' competing constraints are met.

Fidelity. Fidelity is an application defined concept that refers to a notion of quality. The key observation is that different applications can tolerate different levels of quality across the different dimensions of its definition. For example, an application may optimally sample at a frequency of once per minute but still be able to function if this sample rate drops to once per five minutes. A variety of functions preformed in a sensor network setting can be run in a multi-fidelity setting. For example, sensing fidelity can be modified by sampling at different frequencies. Similarly, storage fidelity can be modulated by varying the storage format of data or the granularity at which information is stored (e.g., raw information can be stored for high fidelity while averages may suffice for lower fidelity).

To create a framework for supporting resource sharing among concurrently executing applications, it is necessary to first determine if the resource availability is such that all applications can be supported at their optimal levels of quality. If not, then the framework must use application-specified fidelity tolerances to drive a policy engine for resource allocation and sharing.

Consider an example of a sensor network where all hosts combined have CPU and battery resources of 250 units (cycles/sec in million) and 315 units (Microjoule) available respectively as shown in Table I. There are two applications that want to run in this network. The first usually runs in the background to perform low-level target detection of a mobile intruder. It requires 210 units of CPU cycles and 240 units of energy to achieve 90% fidelity (assuming fidelity is proportional to resource demand and there is no discernible difference running at 90% instead of 100% fidelity). It is willing to tolerate a decrease in fidelity, but only down to 60% (the sensor may switch to sleep mode more frequently thereby reducing the sensing and communication overhead), at which point it requires 140 units of CPU cycles and 160 units of energy to perform its required sensing and communication tasks. The second application measures the current visibility in the region of interest. It requires 100 units of CPU and 150 units of energy to operate at 80% fidelity. It will tolerate a decrease in fidelity, down to 60%, at which point it requires at least 80 units of CPU and 120 units of energy. To enable these applications to execute concurrently and ease the tensions between their resource requirements, we need to investigate the optimal allocation of resources and tolerate some degradation of the fidelity of each individual application. For example, the first application can be executed with

fidelity of 70% (168 units of CPU and 192 units of energy) and the second with fidelity 60% (80 units of CPU and 120 units of energy) resulting in a total resource usage of 248 units of CPU cycles and 312 units of energy. Different combinations of this fidelity level and resource allocation are possible depending upon the available network resources. For simplicity, we do not delve into different dimensions of each individual application fidelity in this example although we characterize this in Section III.

	CPU cycles/sec (million)	Energy in μJ	Fidelity (ρ)
Network	250	315	
App1	210	240	ρ_{max} = 90%
App2	100	150	ρ_{max} = 80%
App1	140	160	ρ_{min} = 60%
App2	80	120	ρ_{min} = 60%
App1	168	192	$\rho_{adaptive}$ = 70%
App2	80	120	$\rho_{adaptive}$ = 60%

TABLE I
ADAPTIVE MULTI-APPLICATION SCENARIO

Our goal is to enable the automatic determination of such resource allocations in a general way that best satisfies the applications' fidelity requirements (e.g., data timeliness, data quality, location of sensing) and best utilizes the available network resources by maximizing the number of concurrent applications. To that end, we discuss fidelity-aware resource allocation and application selection to support multiple concurrent applications in wireless sensor networks. Our main contributions are the following.

- **Multi-Fidelity based Resource Decomposition (MFRD):** We describe a scheme to specify application requirements in a fidelity spectrum. This provides a principled way to degrade fidelity on the spectrum (Section III) as the network's resource constraints are considered.
- **Multi-Applications Task Sharing (MATS):** Based on applications' task and fidelity breakdowns, we show how we can identify tasks to be shared across applications. To accomplish this, we consider each application's minimum task and fidelity requirements and maximize the task sharing across applications (Section IV).
- **Utility-based Application Selection (UAS):** We present a linear time search heuristic algorithm for determining the concurrent applications that can be accommodated with the fidelity requirement within the available network resources (Section V).

In this paper, we focus on the problem of determining the best possible fidelities that different applications can be supported at, given the total available resources in the network. We do not focus on the task assignment problem—assigning the tasks comprising these applications to nodes in the network. The solution to this problem may require further constraining the applications that can be supported or further reducing the fidelity with which they can be supported. This will be investigated in future work. We evaluate our approach in a simulator and demonstrate the performance of our system (Section VI). Our approach is a first step in making sensor networks more general and flexible in their support of a priori unknown application constraints.

II. OUR SYSTEM COMPONENTS

In this section, we outline and discuss the different components of our system model as shown in Fig. 1. In the following sections, we describe each of these components in greater detail.

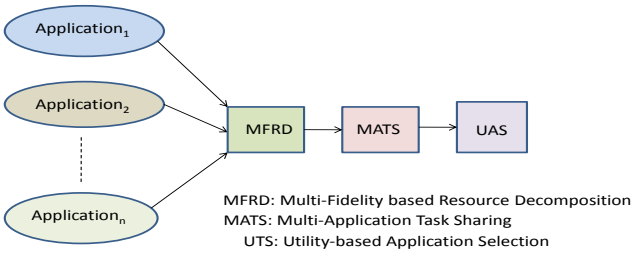

Fig. 1. System Components

Multi-Fidelity based Resource Decomposition: We present a model where fidelity-aware applications can be tailored to execute at different granularities of fidelity for different associated costs in terms of resource utilization. We assume that applications typically prefer high fidelity behavior but can tolerate results of lower fidelity when the resources are limited. We also assume that achieving higher fidelity requires greater resource utilization in comparison to low fidelity applications. We represent each application's fidelity with multi-dimensional metrics that capture the relationship between fidelity and resource consumption. In addition, we define a performance metric index that captures the relationship between an application's requests for resources and the available network resources.

Multi-Application Task Sharing: A key observation in our approach is that maximizing the sharing of tasks among multiple applications eases the accommodation of concurrent applications in a network. To determine the best allocation of resources to potentially competing application tasks, the allocation algorithms must consider not only the fidelity of the applications but also a way to maximize task sharing across all the applications. To that end, we investigate a multi-applications task sharing scheme based on a task dependency graph.

Utility-based Application Selection: To maximize the number of concurrently executing applications, we have formulated a utility based application selection problem. We define a system utility function and propose a constraint-optimization problem based on the individual application's fidelity and resource constraints. We use this system utility function and the degree of task sharing across the spectrum of different applications to determine a correct subset of applications that can run on the network.

III. MULTI-FIDELITY BASED RESOURCE DECOMPOSITION

We begin by discussing three different applications related to an example target detection problem. A network of sensors, cameras, and servers can be used to satisfy the diverse needs of those applications. For example, Custom and Border Patrol personnel may need real-time data to see if a detected target is a potential threat. Homeland security personnel may ask for high fidelity results if there is a high risk of threat or potential match of a target with existing federal database. The project's support personnel might be interested to see the intermediate output of the system and how the performance of the system could be improved. In general, different applications have different expectations of the fidelity metric (delay in this example) in receiving updates.

As can be seen, one of the major challenges in wireless sensor networks is how to enable multiple applications with varying fidelity requirements to execute concurrently in a network with limited resources. One option is to allow those applications to execute independently without sharing any task or network resources, but that would result in wastage of network resources. Additionally, it may not be possible to satisfy all of the applications' requirements given the limited nature of sensor network resources. An alternate approach is to allow applications to share their tasks and resources jointly and if needed to reduce the fidelity of an application to maximize the number of concurrently executing applications.

In this section, we discuss such a multi-fidelity based approach to vary applications' fidelities to accommodate multiple concurrent applications. The intuition is that an application can be satisfied with different fidelity levels by varying its resource request; the higher the resources allocated to it, the better the fidelity. We now discuss how to go about defining multi-fidelity applications in greater detail.

A. Application Task and Fidelity Specification

Different applications expect sensor networks to perform a variety of tasks from three categories: *capture, storage,* and *distribution*. Each of these tasks can be accomplished at various fidelity levels. We now provide example fidelity metrics that can be associated with each of these tasks. These examples are are not intended to be exhaustive but illustrative.

In the *capture* category, the sensor network can perform a variety of operations such as: *collect data from on board sensors*; *cooperate to perform distributed sensing tasks*; *aggregate temporally*; *aggregate spatially, etc.* Each of these operations can be captured concretely with a numerical fidelity measure. For example, data sampling can be represented at a variety of frequencies (e.g, 8, 16, or 24 kHz). Other measures relating to the fidelity of the sampling task can be similarly specified. Link latency can be measured in milliseconds (e.g., 15, 30, 45, 60 ms). Similarly, the image format can be represented as a function of the scheme such as JPEG, BMP, GIF, or PNG etc.

In the *storage category*, the sensor network and its composite nodes can perform the following types of tasks: *store data locally*; *handle load balancing, replication, and redundancy*. Once again, this task can have fidelity representation along several dimensions. For example, *data compression ratio* can be used to interpret the impact of the compression algorithm (LWZ, ZIP, JPG etc.) *Data freshness* may alternatively be used to describe how often its is possible to receive a new measurement.

The final category in our framework is *distribution*, which handles the communication of data among nodes in the network. Aspects relating to the fidelity of distribution can once again refer to a variety of operations like, *gaining access to the medium*, *creating and using multihop routes*, and *creating and maintaining distributed data structures*. Some examples of fidelity metrics that describe these operations are, the *maximum packet loss due to collision*, the *expected packet loss* and *data timeliness*. We now discuss how to formally capture the variety of fidelity levels that these operations can operate in.

B. Application Fidelity Decomposition

We begin by defining a mathematical representation for a multi-fidelity application. An application's fidelity changes dynamically as its resource allocation changes based on competing applications. We assume $F_{i1}, F_{i2}, \ldots, F_{in}$, are the fidelity dimensions of the i-th application, and we denote the possible fidelity vectors by $F_i = F_{i1} \times F_{i2} \times \ldots \times \ldots F_{in}$. Thus, we define an application's fidelity as a multi-dimensional vector and envision that the application can tolerate lower fidelity in certain dimensions if there are insufficient resources to achieve higher fidelity levels. For a handful of the example metrics described previously, we show briefly how we express them in terms of a multi-dimensional utility function.

Data Sampling Rate: The data sampling rate of sensors can vary from 8 kHz (low quality) to 72 kHz (high quality), and each setting can provide a different level of fidelity.

Sampling rate	8	16	24	48	72	...
Fidelity(F_{i1})	ρ_1	ρ_2	ρ_3	ρ_4	ρ_5	...

TABLE II
SAMPLING VS. FIDELITY

Data Compression Ratio: Different compression techniques can be used such as LZW compression, ZIP compression, lossy JPG compression etc. We use compression ratio to quantify the reduction in data representation size produced by a data compression algorithm.

Compression Ratio	1	2	4	8	16	...
Fidelity(F_{i2})	ρ_1	ρ_2	ρ_3	ρ_4	ρ_5	...

TABLE III
COMPRESSION VS. FIDELITY

Data Timeliness: This is the end-to-end delay of routing the data. We assume that it varies from 20 ms to 100 ms.

Delay (ms)	100	80	...	60	40	20
Fidelity(F_{i3})	ρ_1	ρ_2	...	ρ_3	ρ_4	ρ_5

TABLE IV
DELAY VS. FIDELITY

C. Application's Utility

For ease of defining the utility function we consider an ordering of an application's individual fidelity dimensions. We map each of the above fidelity dimensions to a non-decreasing multi-dimensional utility function $u_{ij}(.)$ and represent a dimension-wise fidelity utility function as follows. $F_{i1} \propto u_{i1}(\rho_{ij})$; $F_{i2} \propto u_{i2}(\rho_{ij})$; $F_{i3} \propto u_{i3}(\rho_{ij})$ ... etc. Thus the utility of an application can be expressed as $u_i = \sum_{j=1}^{k} u_{ij}(\rho_{ij})$ where k is the k-th dimension of the fidelity vector.

D. Performance Index

Using a multi-fidelity representation is useful because the network can have a varying degrees of resources available at different times. To capture, the availability of resources for each application we define a parameter called the performance index metric, α. To determine the network's available resources at runtime, we probe the network in specific time intervals to get the network load statistics and make predictions on resource availability. For the ease of modeling, we assume that this gives us an estimate of the CPU cycles available (A_{cpu}), the available network bandwidth (A_{bw}), and the available energy (A_{en}). There

are other methods to estimate the resources available [6], and perfecting this estimation is left for future work.

To determine the resource request of the i^{th} application, we define a relation between resource request R_i and the application's achievable fidelity F_i such that $r \xrightarrow{i} \rho$. That is, for certain combinations of different resources r the application can achieve its fidelity point ρ. One application might increase the data sampling rate over time to decrease spatial data aggregation. Increasing the data sampling rate may result in increased energy depletion, but decreasing the spatial aggregation could reduce the communication overhead (network bandwidth) and link latency. In both cases, the application asks for different system resources but satisfies the same fidelity requirements. We assume an application's resource request is presented in terms of CPU cycles (R_{cpu} units), network bandwidth (R_{bw} units), and energy (R_{en} units) required to achieve the fidelity point ρ. Finally, we define the performance index as a function of the application's resource request and the network's available resources. For the i-th application we define it as follows: $\alpha_i = \frac{R_{cpu}}{A_{cpu}} + \frac{R_{bw}}{A_{bw}} + \frac{R_{en}}{A_{en}}$.

IV. Multi-Application Task Sharing

In this section we present a multi-application task sharing approach to minimize the resource usage across multiple applications that have commonalities among their component tasks.

Affiliation Set: We define an affiliation set, $\mathcal{A}$, as a choice of the subset of tasks for a specific application that, when performed, meets the application's minimum fidelity (ρ_i^{min}) requirement. Let $r(t)$ be the resource request associated with task t. Then $R(\mathcal{A}) = \bigcup_{t \in \mathcal{A}} r(t)$, where $R(\mathcal{A})$ denotes the total (maximum) resource request of an affiliation $\mathcal{A}$ to satisfy the application's minimum fidelity requirement.

Let us consider different concurrent applications having different affiliation sets as shown in Fig. 2. There may exist resource dependencies between the common tasks. For example, the set of tasks in one affiliation set may produce the aggregated data which may be required by another affiliation set. Again the tasks in one affiliation set which already capture the data, can store it for being retrieved later by the set of tasks in another affiliation set for a different application. We represent these relationships using dependency graphs. Each node in the dependency graph represents a local affiliation, and an edge between two nodes implies that each affiliation shares some tasks in common. The weight of an edge represents the resource utilization of the common tasks in this set of common tasks that the edge represents. Thus the dependency graph can be represented as a weighted graph, $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ where the nodes in $\mathcal{V}' = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k\}$, and an edge $e' = (\mathcal{A}_i, \mathcal{A}_j) \in \mathcal{E}'$, if and only if $\mathcal{A}_i \cap \mathcal{A}_j \neq \emptyset$. Each node $\mathcal{A}_i \in \mathcal{V}'$ represents a local affiliation set meeting the application's objective. The weights on the edges and nodes are defined as follows:

$w(e') = \bigcup_{t \in \mathcal{B}} r(t)$ where $e' = (\mathcal{A}_i, \mathcal{A}_j)$ and $\mathcal{B} = \mathcal{A}_i \cap \mathcal{A}_j$

We also define the weight of a node or the degree of an affiliation $\mathcal{A}$ as the sum of weights of all the edges from a node: $d_i = \bigcup_{e' \in \mathcal{E}' \text{and imposed on} \mathcal{A}} w(e')$, which represents the *degree* of an affiliation $\mathcal{A}_i$. This gives us an estimate of the task sharing an affiliation set $\mathcal{A}$ may have for a particular application in the presence of multiple concurrent applications.

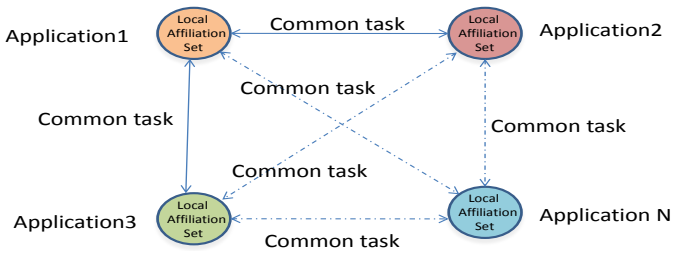

Fig. 2. Mapping of multiple concurrent applications to Affiliation Sets

We use the degree of an affiliation set to prioritize the application execution across multiple concurrent applications. The priority of an affiliation $\mathcal{A}$ depends upon its degree, as a higher degree corresponds to larger sharing of tasks across applications. Thus a higher degree of an affiliation means a better chance of satisfying more concurrent applications due to a larger overlap of common tasks. We consider this degree information later in our utility-based application selection heuristic algorithm.

V. Utility-based Application Selection

In this section, we propose a heuristic search algorithm to determine the optimal subset of the applications that can be accommodated given the applications' fidelity requirements and resource limitations of the network.

A. System's Utility Determination

We define an application's utility based on the fidelity-utility function, where the application utility is directly proportional to the fidelity value. Considering the multi-application scenario, we define the overall system utility as follows:

$$S(\rho_1, \ldots, \rho_n) = \sum_{i=1}^{n} w_i u_i(\rho_i) \tag{1}$$

where w_i is a weighting factor ($0 \leq w_i \leq 1$), which represents the relative importance of the applications.

Problem Definition: Determine the optimal subset of the applications, given the fidelity and resource restrictions, such that the overall system utility is maximized. Thus we can define the optimization problem as follows:

$$\begin{aligned} &\textbf{maximize} && S(\rho_1, \ldots, \rho_n) \\ &\textbf{subject to}: && \rho_i \geq \rho_i^{min} \quad i = 1, \ldots, n, \quad \text{(Fidelity Constraints)} \\ & && \sum_{i=1}^{n} \alpha_i \leq 1 \quad \text{(Resource Constraints)} \end{aligned} \tag{2}$$

B. Choice of the Applications

The optimization problem stated in Equation 2 can be solved if all the required parameters are known a priori. But considering the dynamic nature of sensor networks, it is nearly impossible to have complete knowledge of the operating conditions. Thus we design a greedy heuristic algorithm to solve it.

Greedy Algorithm: We describe a simple greedy heuristic algorithm in the absence of any prior knowledge of the network. Our goal is to determine the optimal subset of applications (called θ) that maximizes the overall system utility in

addition to satisfying the minimum fidelity requirement ρ_i^{min} for each of the application. Clearly, one solution is to iterate through all possible combinations, computing system utility $\widehat{S(.)}$ for each combination of applications. However, for efficient operation, we have developed a selection heuristic that is instead linear in the number of applications.

The heuristic is based on the observation that the increase or decrease in system utility (based on Equation 1) of adding an application A_i to an existing set θ is dependent on the nature of the specific utility function, the degree of task sharing d_i, and the performance index factor α_i. A higher value for this sharing term indicates a greater preference for selecting an application (ideally, an application with increasing d_i would reduce the overall performance index α_i in the presence of other concurrent applications). A lower value of α_i also indicates lower resource consumption. Accordingly, our heuristic sorts the set of available applications, B, in descending order of this term, increments the individual application's fidelity using a step function until the total performance index factor exceeds the limiting value 1 and keeps adding applications to θ until the total system utility decreases. Fig. 3 shows our heuristic's pseudocode.

```
Procedure Application_Selection(input set B, ρ_i^min, α_i)
1.  Initialize an empty set of applications; θ = φ;
    ρ_i = ρ_i^min; MaxSysUtility = ∞;
2.  Sort the application set B into a list L in
    decreasing order of factor (d_i/α_i);
3.  For (i = 1; i < |B|; i++)
4.      θ = θ + L(i); //set-theoretic addition
5.  Calculate R_i − d_i; //Resource Request - Resource Sharing
6.  Update α_i;
7.      Compute performance index factor Σ_{i=1}^n α_i for θ
8.      if (Σ_{i=1}^n α_i < 1)
9.          ρ_i = ρ_i + (ρ_i^max − ρ_i)/2; // Increment fidelity
10.     Compute system utility Ŝ(.) for ρ_i
11.     if (Ŝ(.) − MaxSysUtility) < 0
12.         break;
13.     else MaxSysUtility = Ŝ(.)
14. End-For
15. return {θ, ρ_i, Ŝ(.)}
```

Fig. 3. Application Set Selection Heuristic

VI. Evaluation

In this section, we evaluate the feasibility of the ideas discussed in this paper. We wrote a Java based application to do performance analysis based on typical use cases that a user of our approach is likely to undertake. We simulated a setting where there are ten applications that would like to run simultaneously on a given sensor network. Each application is broken into a random number of sense, store, and communication tasks and is assigned a random of resource requests for each of these tasks (typically a random number between 10 and 100 units). In addition, the simulator randomly determines the number of tasks that are shared between applications (i.e., the degree of the Affiliation Sets). The amount of resources available in the network is generated randomly for each run. These are typically a random fraction of the resources requested. Every application is assigned a utility function that allows it to generate a range of fidelities that the application can operate under as discussed earlier.

Our first experiment was designed to establish whether having multi-fidelity applications is beneficial to creating sensor network platforms. Fig. 4 demonstrates that this is indeed the

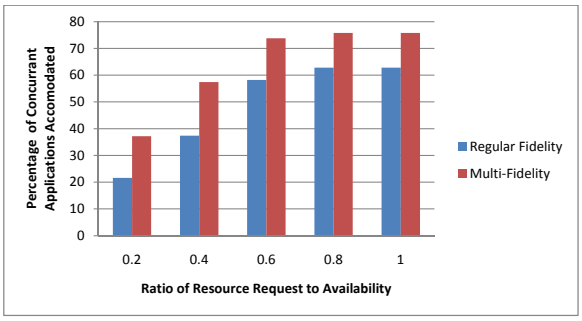

Fig. 4. Effectiveness of Multi-Fidelity Representation

case. The X-axis represents the ratio of resources requests to the actual availability on average. The Y-axis represents the percentage of all applications that can be run simultaneously in the network for two different configurations. The first configuration (blue bar) represents traditional applications that come with only one fidelity level. These applications can either run at that fidelity level or not at all. The second bar (red bar) represents applications that are designed using the ideas in this paper (i.e., the applications are *fidelity-aware*). Here, applications run at several fidelities and we can lower the fidelities of some applications to satisfy more applications in the sensor network concurrently. As can be seen with the graph, writing applications that can be run at several fidelity levels is greatly valuable. We can use the same sensor network to more efficiently accommodate more applications simultaneously. In our specific experimental settings, we typically observe that we can accommodate two extra applications (out of the ten possible applications) when employing multi-fidelity applications.

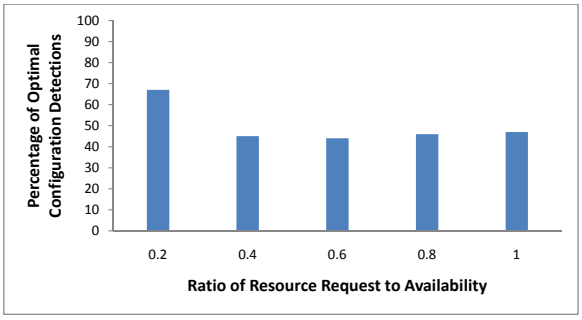

Fig. 5. Effectiveness of Greedy Algorithm

Our second experiment assessed the effectiveness of our algorithm in detecting a good combination of applications to schedule on a sensor platform. Given the large number of applications that can be run in a variety of fidelity settings, the goal of our algorithm is to pick a good combination of applications and their corresponding fidelity setting. We call the set of chosen applications and their corresponding fidelity settings a configuration. Fig. 5 demonstrates that our greedy algorithm is quite effective in practice. The X-axis, once again, represents the ratio of resource requests to availability. The Y-axis highlights the percentage of times our algorithm achieved an optimal configuration out of 100 runs. By this we mean, the percentage of times where resources in the network was used most efficiently. That

is, there was no better configuration of applications that could have used the resources in the network more efficiently. As can be seen from the graph, the percentage of times that we discover the optimal set is around 40%. This is a pretty high number given the large number of combinations that are possible taking into account all the applications and the various fidelities that they can exist in. We believe this is because the degree of task sharing is an important metric in making our greedy algorithm effective. This experiment demonstrates that relying on a heuristic and employing a greedy algorithm works well in practice and more often than not, finds a good solution in a large search space.

Figs. 4 and 5 demonstrate that it is beneficial to treat a sensor network as a platform and run several applications simultaneously on it. However, to use the platform effectively, applications need be written so as to run in several fidelity settings. An algorithm, such as our greedy algorithm, can then be used to select the right configuration at run time.

VII. Related Work

Our work relates to several areas of research explored in wireless networks and other areas. In this section, we place our work in context. The idea of using multiple fidelities for individual applications has been discussed previously in the literature. Satyanarayanan *et al.* [7] formally define the notion of multi-fidelity algorithms as those that allow a range of possible outcomes instead of a single output specification. They established the usefulness of using multi-fidelity approaches to interactive applications in particular [6]. The approach taken in this paper is inspired by this work, and we explore the implications of using multi-fidelity algorithms in the context of a sensor networks platform. In addition our approach borrows from the notion of fidelity vectors introduced in [3]. Another theme that runs in this work is the ability to break down applications into tasks and share tasks while running applications concurrently. The Abstract Task Graph [1] is an effort in a similar vein. The user specifies a set of abstract data items and tasks that are performed by the sensor network. Our breakdown of sensor network tasks into sensing, storage, and communication is in the same spirit. However, our focus is on using such a task breakdown for application sharing, thereby enabling a sensor network platform supporting multiple concurrently executing applications. There has been some work on running multiple applications in a sensor network in the database community [2], [8]. Such work has typically focused on intelligently sharing some sensor network tasks such as reusing aggregated data and using the same data for multiple applications. Our work exploits some of these observations and also focuses on sharing network resources like bandwidth and energy. Finally, we discuss other efforts relating to deciding which applications to schedule in a sensor network platform. An analytic evaluation of satisfying the QoS needs of multiple applications along multiple QoS dimensions was presented in [3] for real time systems. Given the difficulty of knowing the exact state of the environment and the resources available, we believe that analytic solutions will be difficult to employ in sensor networks. Consequently, we employed a utility driven approach. A similar technique was studied for large radio networks in [4]. Several other utility-based approaches such as resource allocation in sensor networks SORA [5] have also been studied. These rely on reinforcement learning and economic models for energy optimization in sensor networks instead of the greedy heuristic employed in our work. In future work, we plan to look at all the possible options—analytic, heuristic, learning, optimization etc. and understand the tradeoffs of using each to determine what set of applications to run concurrently in a sensor network.

VIII. Conclusion and Future Work

We have presented a new way of looking at application selection and resource sharing in sensor networks, specifically addressing the needs of sensor networks that support multiple differing and potentially competing applications. Our approach introduces a fidelity function to judiciously trading off application's fidelity decay in presence of multiple applications and limited network resources. We also formulate a constraint-optimization problem and propose a greedy heuristic algorithm to select the optimal subset of concurrent applications within the limited fidelity and resource budget.

In future work we plan to address three issues. First, we plan to explore the software engineering implications of writing applications that run at several fidelities. Can applications that are currently written be easily changed so as to run at different fidelities. Can we develop design patterns to ease this effort? Second, we will investigate both analytic solutions to our constraint optimization problem and learning based techniques that should perform better than our greedy heuristic presented here. Finally, we investigate the impact of error on our perception of resources. In this work, we assumed the we had real-time knowledge of all the resources available in the network. This is clearly unrealistic in actual deployment. We will investigate the impact of having imperfect information about resource usage and consumption on our algorithms.

References

[1] A. Bakshi, V. K. Prasanna, J. Reich, and D. Larner. The abstract task graph: a methodology for architecture-independent programming of networked sensor systems. In *EESR '05: Proceedings of the 2005 workshop on End-to-end, sense-and-respond systems, applications and services*, pages 19–24, Berkeley, CA, USA, 2005. USENIX Association.

[2] R. Huebsch, M. Garofalakis, J. M. Hellerstein, and I. Stoica. Sharing aggregate computation for distributed queries. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 485–496, New York, NY, USA, 2007. ACM.

[3] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, and J. Hansen. A scalable solution to the multi-resource qos problem. In *Proceedings of the 20^{th} IEEE Real-Time Systems Symposium*, pages 315–326, 1999.

[4] M. Li, T. Yan, D. Ganesan, E. Lyons, P. Shenoy, A. Venkataramani, and M. Zink. Multi-user data sharing in radar sensor networks. In *Proceedings of the 5^{th} International Conference on Embedded Networked Sensor Systems*, pages 247–260, 2007.

[5] G. Mainland, D. C. Parkes, and M. Welsh. Decentralized, adaptive resource allocation for sensor networks. In *Proceedings of the 2^{nd} conference on Symposium on Networked Systems Design & Implementation*, pages 315–328, 2005.

[6] D. Narayanan and M. Satyanarayanan. Predictive resource management for wearable computing. In *Proceedings of the 1^{st} International Conference on Mobile Systems, Applications and Services*, pages 113–128, 2003.

[7] M. Satyanarayanan and D. Narayanan. Multi-fidelity algorithms for interactive mobile applications. *Wireless Networks*, 7(6):601–607, 2001.

[8] N. Trigoni, Y. Yao, A. Demers, and J. Gehrke. Multi-query optimization for sensor networks. In *Proceedings of the 1^{st} International Conference on Distributed Computing in Sensor Systems*, pages 307–321, 2005.