Energy-Efficient Task Mapping for Data-driven Sensor Network Macroprogramming

Animesh Pathak, *Member, IEEE,* and Viktor K. Prasanna, *Fellow, IEEE.*

Abstract—

Data-driven macroprogramming of wireless sensor networks (WSNs) provides an easy to use high-level task graph representation to the application developer. However, determining an energy-efficient initial placement of these tasks onto the nodes of the target network poses a set of interesting problems. We present a framework to model this task-mapping problem arising in WSN macroprogramming. Our model can capture placement constraints in tasks, as well as multiple possible routes in the target network. Using our framework, we provide mathematical formulations for the task-mapping problem for two different metrics — energy balance and total energy spent. For both metrics, we address scenarios where a) a single or b) multiple paths are possible between nodes. Due to the complex nature of the problems, these formulations are not linear. We provide linearization heuristics for the same, resulting in mixed-integer programming (MIP) formulations. We also provide efficient heuristics for the above. Our experiments show that our heuristics give the same results as the MIP for real-world sensor network macroprograms, and show a speedup of up to several orders of magnitude. We also provide worst-case performance bounds of the heuristics.

*Index Terms—***Sensor Networks, Task-Mapping, Macroprogramming.**

I. INTRODUCTION

APPLICATIONS executing on parallel and distributed systems can often be represented as tasks running on the constituent nodes of the system, interacting to produce the result. The efficient mapping of these tasks to the system nodes is a well-studied problem in classical parallel and distributed computing research. Wireless Sensor Networks (WSNs) are rapidly emerging as a new class of distributed system, with features that are different from traditional systems. Various high-level programming abstractions have been proposed to assist in application development for WSNs. *Data-driven macroprogramming* [1] refers to the general technique of specifying the WSN application from the point of view of data-flow. In sense-and-respond applications such as traffic management [2], building environment management [3], target tracking [4] etc., the system can be represented as a set of tasks running on the system's nodes – producing, processing and acting on data items or streams to achieve the system's goals. The motivation of our work (explained further in Section II-A) comes from the fact that WSNs have certain interesting properties (large scale, heterogeneity, energy-deficient nodes, etc.) which make the mapping of these tasks onto the nodes of the underlying system (details of which are known at compile time) an important part of the compilation of the macroprogram, and open the way for optimizations to be

A. Pathak and V. K. Prasanna are with the University of Southern California.

performed at this stage to make the resulting WSN more efficient.

We note that task mapping in this context differs from the traditional task-mapping problems seen in parallel and distributed computing in several aspects:

1) The task graph here is *constrained* in the sense that some tasks have a one-to-one correspondence with the nodes in a system, while the placement constraint of others may not be as strict. For example, a temperature sampling task can be placed only on nodes with temperature sensors attached to them, while the task that computes the average of temperature readings in a room has much relaxed mapping constraints.

2) Often in the classical task-mapping scenarios, tasks are assumed to be independent of each other and do not communicate. In cases where they do, a point-to-point link is usually assumed between all nodes. In the case where routing is involved, the intermediate nodes only introduce delays, but are not affected otherwise. On the other hand, the WSN applications studied by us consist entirely of communicating tasks. This communication of data between tasks on different nodes in a WSN affects other nodes in the system as well, since the nodes involved in routing also spend energy in the process.

3) In cases where routing is involved, classical task-mapping algorithms either have full control over routing, or assume a specific routing. Since our goal is for the macroprogramming framework to be modular, our techniques do not assume a certain routing protocol. Instead, our modeling framework allows an interface for developer to specify certain facts about the routing protocol being used.

4) While the most common constraint in traditional parallel and distributed systems is *latency*, i.e., the time taken for the tasks to complete execution, most sensor networking applications are designed to sense-and-respond for long periods of time. On the other hand, metrics such as *system life time* and *energy spent* at the node and system level are much more important in WSNs. We focus on two measures of energy-efficiency in this paper.

Additionally, although the initial information (positions, energy levels) about the target nodes is known, during the lifetime of the WSN, changing conditions, either external (variations in the environment) or internal (nodes running out of energy) may change the circumstances. Our algorithms do not address these unpredictable situations, and instead aim to provide a "good" initial mapping of tasks. We assume that during the lifetime of the system, remapping of tasks will

occur to face these circumstances, for example, a distributed task-remapping algorithm can be triggered when the energy at any node goes below a certain fraction of its initial energy level. Our work attempts to utilize the global knowledge available at compile-time to obtain efficient results.

We introduced the mathematical framework for solving such a task-mapping problem in [5], and briefly summarized a mathematical formulation and some heuristics for the same. This paper extends the work with a detailed description of our model and the techniques used to solve the task-mapping problems, in addition to addressing the case when multiple routes are possible between nodes. Overall, we make the following **contributions**:

- In Section II we provide a framework to model the problem of task-mapping for data-driven sensor network applications, with tasks subject to placement constraints and channels annotated with data-rates.
- In Section III we propose a mixed integer programming (MIP) formulation to obtain task mappings in order to optimize for the energy balance and total-energy minimization goals, both in the cases where a single route is available between each pair of nodes. Since the formulation is non-linear, we provide substitution-based techniques to linearize the MIPs.
- Although the MIP formulations give optimal results, they may take inordinately large times to terminate for large real-world scenarios. In Section IV, we provide greedy heuristics for the two problem instances, along with their worst-case performance analysis.
- In Section V, we provide formulations of the above problem instances when there are multiple routes available between each pair of nodes in the system. We provide (linearized) MIP-based and greedy techniques to solve these generalized problems.

Our experimental results, discussed in Section VI, show the performance comparison between the techniques, using realistic applications and deployment scenarios. Our greedy heuristics are shown to obtain the optimal solution for most of these scenarios, while gaining significant speedups over the MIP technique. We discuss related work in Section VII and conclude in Section VIII.

II. PROBLEM FORMULATION

A. Motivation

As an example of data-driven macroprogramming representation, consider the following (simple) application – A room is instrumented with six wireless nodes, with three nodes equipped with temperature sensors, and two nodes connected to actuators for controlling the temperature of the room. We need to periodically determine the average temperature in the room, compare it with a threshold, and produce the corresponding actuation. One way of designing such an application at a high-level using a data-driven approach is shown in the top part of Figure 1 (Note that the task graph for complex applications can be an arbitrary directed acyclic graphs). Tasks T_1, T_2 and T_3 are temperature sampling tasks, which fire at a rates of f_1, f_2, f_3 and generate ambient temperature readings

of size s_{14}, s_{24}, s_{34}. Task T_4 calculates the average of these readings and feeds it to T_5, which determines the action to be taken. Tasks T_6 and T_7 act upon the data generated by T_5, and control the actuators. The system for which this application is being designed is shown in the lower part of the same figure. The nodes equipped with temperature sensors are marked with a **T**, while the ones equipped with actuators are marked with an **A**. The mapping of tasks T_1 through T_7 onto the nodes
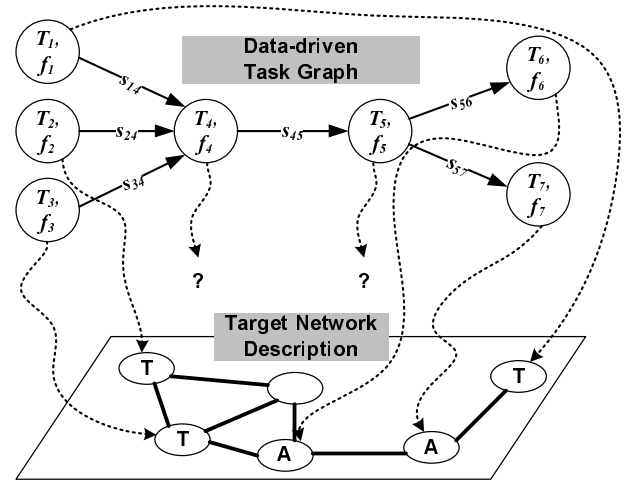

Fig. 1. Temperature management application - Task graph and target network description.

of the target network is an instance of the problem faced while compiling data-driven macroprograms for WSNs. The placement of the sensing tasks (T_1, T_2, T_3) and the actuating tasks(T_6 and T_7) are pre-determined to the nodes with the relevant capabilities. This fact is shown using curved broken lines in the figure. However, tasks T_4 and T_5 can be placed on any of the nodes in the floor, thus allowing for optimizations in this process.

Our aim is to capture the following aspects of the problem in a mathematical formulation:

- The data flow between tasks
- The different firing rates of the tasks
- The placement constraints of tasks onto system nodes
- The heterogeneity between the system nodes, both in terms of their initial energy capacities, as well as their ability to host certain tasks
- The heterogeneity between the various network links in the target system, in terms of energy spent per unit of data transmitted
- The energy spent at the nodes during sensing, computation, and communication.

B. Application and System Model

A **Network Description** N represents the target system of physical nodes where the WSN application is to be deployed. Each node k $(k = 1, \ldots, n)$ has the following properties:

- its initial energy reserve e_k^0. We assume that the system operates in *rounds*, and denote the energy remaining at node k after t rounds by e_k^t. A **round** is defined as the

least time-period after which the system behavior repeats itself.

A **Data-driven Task** i represents the sensing, processing or actuation activity in a WSN, with the following properties:

- its firing rate f_i, denoting the number of times it is invoked in one round. For tasks that are not necessarily invoked in a regular manner, developers can determine the firing rates using probabilistic estimates.

A **Data-driven Task Graph** $D = (DT, DE)$ is a directed acyclic graph (DAG) consisting of the following:

- A set $DT = \{1, \ldots, i, \ldots, m\}$ of data-driven tasks.
- A set $DE \subseteq DT \times DT$ of edges. Each edge (i, j) is labeled with the size s_{ij} of the data that task i produces for task j upon each invocation.

The **Task Execution Energy Matrix** $\mathcal{T}$ is an $m \times n$ matrix, where $\mathcal{T}_{ik}$ denotes the energy spent by node k per invocation of task i, if i is mapped onto node k. $\mathcal{T}$ can also be used to specify placement constraints as in the figure above, by setting the value of $\mathcal{T}_{ik}$ to ∞ in cases where task i cannot be instantiated on node k.

The **Routing Energy Cost Matrix** $\mathcal{R}$ for N is a $n \times n \times n$ matrix, with $\mathcal{R}_{\beta\gamma k}$ denoting the energy consumed per unit of data at node k while routing messages from node β to γ. Since the task mapping algorithms do not control the routing of data between nodes, $\mathcal{R}$ provides an estimate of the energy spent in routing.

The **Task Mapping** is a function $M : DT \rightarrow N$, designating task i to be placed on node $M(i)$.

At this moment, we would like to acknowledge that the above system model includes information that may not be available in all practical situations. The routing energy cost matrix $\mathcal{R}$, especially, may not be available for unplanned ad-hoc WSN deployments, and may indeed change during the operation of the system depending on ambient conditions. However, we believe that there are a large number of application scenarios, especially where office, home, or industrial buildings are instrumented with sensors and actuators, where this model might be valid. In systems where the costs will change with time, the basic model still holds, and can be used to design efficient distributed algorithms for task migration. We summarize the symbols used in our model in Table I.

C. Energy Costs

In a sensor network, the *cost* that developers are largely concerned with is the *energy spent* by the nodes as the system operates. We therefore use the terms *cost* to mean the energy spent at a node throughout this paper, unless otherwise stated. Using the model defined above, we compute the following costs[1].

Computation Cost: At each node $k \in N$, the computation cost in each round is given by

$$C_{\text{comp}}^k = \sum_{i:M(i)=k} f_i \cdot \mathcal{T}_{ik} \qquad (1)$$

---

[1] Note that the cost of sensing is included in the $\mathcal{T}_{ik}$ of the sensing tasks.

Symbol	Meaning
$N = \{1, \ldots, n\}$	Set of nodes in the system (indexed by k)
e_k^0	Initial energy at node k
n	Number of nodes in N
$D = (DT, DE)$	Data-driven task graph
$DT = \{1, \ldots, m\}$	Set of tasks (indexed by i or j)
m	Number of tasks in DT
$DE \subseteq DT \times DT$	Edges in task graph
f_i	Firing rate of task i
s_{ij}	Size of data transferred from task i to j each time i fires
$\mathcal{T}_{ik}$	Energy spent by node k per invocation of task i, if assigned
$\mathcal{R}_{\beta\gamma k}$	Energy consumed by node k while routing one unit of data from node β to node γ
$M : DT \rightarrow N$	Mapping of tasks from DT to nodes in N
C_{comp}^k	Computation cost at node k per round
C_{comm}^k	Communication cost at node k per round

TABLE I
SYMBOLS USED IN OUR SYSTEM MODEL

Communication Cost: At each node $k \in N$, the energy spent in communicating messages in each round is given by

$$C_{\text{comm}}^k = \sum_{(i,j) \in DE} f_i \cdot s_{ij} \cdot \mathcal{R}_{M(i)M(j)k} \qquad (2)$$

Using these node-level costs, complex system-level metrics can be represented, as discussed below.

D. Performance Metrics

In this paper, we illustrate the use of above modeling framework to optimize two performance metrics. The first is *energy balance*, which we consider to be achieved when the maximum fraction of energy spent by any node in the system is minimized. In other words,

$$\text{OPT}_1 = \min_{\text{all Mappings } M} \max_{k \in N} \frac{1}{e_k^0} \cdot (C_{\text{comp}}^k + C_{\text{comm}}^k) \qquad (3)$$

For systems designed using data-driven macroprogramming, we can assume that the system undergoes a reconfiguration, resulting in re-computation of the task mapping, and migration of tasks, once the current energy e_k^t of any node k goes below a fraction $\alpha (0 < \alpha < 1)$ of its initial energy e_k^0. The time when this happens is called the **Time to Reconfiguration** (TTR) for the task mapping on the sensor network. Since we assume that the system behavior repeats itself in each round, OPT_1 also maximizes the TTR.

The second performance goal we model using our framework is the more classical *total energy spent* in the entire system. Although we believe that energy balance is a better metric to measure the quality of task placement, we use the goal of minimizing the total energy spent in the system to illustrate the modeling power of our framework. In other words,

$$\text{OPT}_2 = \min_{\text{all Mappings } M} \sum_{k \in N} (C_{\text{comp}}^k + C_{\text{comm}}^k) \qquad (4)$$

For each of the two metrics, a *feasible* solution is possible only when all nodes have non-zero energy left at the end of one

round. If there are no mappings possible for which this holds, the task-mapping algorithms should report failure. In addition to the above, our framework can be used to model other application scenarios also, e.g. when multiple paths between two nodes are possible, or when the nodes are free to behave differently in each round.

E. Evaluation Criteria

One area where task graphs describing sensor network applications are different from those traditionally seen in parallel and distributed computing is their use for sensing the environment they are placed in, and reacting to it. This leads to certain commonly observed relationships between the data-rates on the edges of the task graphs. Therefore, while evaluating the algorithms for task mapping the input graph has to be carefully chosen. The worst-case analysis technique of testing algorithms against randomly generated task graphs with arbitrarily chosen inter-task data-rates may identify certain task mapping techniques as inferior, while these techniques may yield very good results in real-world WSN applications. Consequently, it is important that the task graphs used to evaluate these techniques are drawn from actual WSN applications.

III. MATHEMATICAL FORMULATIONS FOR TASK MAPPING ON WSNS

A. Mixed Integer Programming Formulation for OPT_1

To formulate the problem as a mixed integer programming (MIP) problem, we represent task mapping M by an *assignment matrix* X, where x_{ik} is 1 if task i is assigned to node k, and 0 otherwise.

The problem can then be defined as:

Inputs:
- $D = (DT, DE)$: Data-driven Task Graph
- f_i: Firing rate for task i
- s_{ij}: Size of data transferred from task i to j on each invocation of i
- N: Network description
- $\mathcal{T}$: Task execution energy matrix
- $\mathcal{R}$: Routing energy cost matrix

Outputs:
- X: Assignment Matrix. x_{ik} is binary.

Optimization Goal:

$$\text{minimize } c$$

Constraints:

$$\sum_{k=1}^{n} x_{ik} = 1 \text{ for } i = 1, 2, \ldots, m \quad (5)$$

$$\frac{1}{e_k^0} \Big(\sum_{(i,j) \in DE} \sum_{\beta=1}^{n} \sum_{\gamma=1}^{n} f_i \cdot s_{ij} \cdot x_{i\beta} \cdot x_{j\gamma} \cdot \mathcal{R}_{\beta\gamma k} + \\ \sum_{i=1}^{m} f_i \cdot \mathcal{T}_{ik} \cdot x_{ik} \Big) \leq c \text{ for } k = 1, 2, \ldots, n \quad (6)$$

$$x_{ik} \in \{0, 1\} \text{ for } (i, k) = (1, 1), \ldots, (m, n) \quad (7)$$

$$0 \leq c \leq 1 \quad (8)$$

The summation terms in (6) denote C_{comm}^k and C_{comp}^k respectively. The final constraint ensures that the MIP fails if no feasible solution exists. Note that the above is an MIP since c is real whereas x_{ik} are binary integers. Also, it is not a linear program since product terms $x_{i\beta} \cdot x_{j\gamma}$ appear in the constraints.

The above problem can be converted to a linear MIP by replacing each $x_{i\beta} \cdot x_{j\gamma}$ term with a binary variable $y_{i\beta j\gamma}$, and adding the following constraints:

$$y_{i\beta j\gamma} - x_{i\beta} \leq 0 \quad (9)$$

$$y_{i\beta j\gamma} - x_{j\gamma} \leq 0 \quad (10)$$

$$x_{i\beta} + x_{j\gamma} - y_{i\beta j\gamma} \leq 1 \quad (11)$$

This linearization techniques is derived from [6]. Intuitively, constraint (9) denotes that if edge (i, j) is mapped to path $(\beta \to \gamma)$, then task i is mapped to node β. Similarly, (10) denotes the constraint that if (i, j) is mapped to $(\beta \to \gamma)$, then task j is mapped to node γ. Finally, (11) denotes the condition that if task i is mapped to node β, and task j is mapped to node γ, then (i, j) is mapped to $(\beta \to \gamma)$.

B. MIP Formulation for OPT_2

Using our formulation, the objective of solving the problem to minimize the *total energy* spent by the system can be formulated as follows:

Inputs:
- $D = (DT, DE)$: Data-driven Task Graph
- f_i: Firing rate for task i
- s_{ij}: Size of data transferred from task i to j on each invocation of i
- N: Network description
- $\mathcal{T}$: Task execution energy matrix
- $\mathcal{R}$: Routing energy cost matrix

Outputs:
- X: Assignment Matrix. x_{ik} is binary.

Optimization Goal:

$$\text{minimize } \sum_{k=1}^{n} \left(\sum_{i=1}^{m} f_i \cdot \mathcal{T}_{ik} \cdot x_{ik} + \sum_{(i,j) \in DE} \sum_{\beta=1}^{n} \sum_{\gamma=1}^{n} f_i \cdot s_{ij} \cdot x_{i\beta} \cdot x_{j\gamma} \cdot \mathcal{R}_{\beta\gamma k} \right)$$

Constraints:

$$\sum_{k=1}^{n} x_{ik} = 1 \text{ for } i = 1, \ldots, m \quad (12)$$

$$\sum_{(i,j) \in DE} \sum_{\beta=1}^{n} \sum_{\gamma=1}^{n} f_i \cdot s_{ij} \cdot x_{i\beta} \cdot x_{j\gamma} \cdot \mathcal{R}_{\beta\gamma k} + \\ \sum_{i=1}^{m} f_i \cdot \mathcal{T}_{ik} \cdot x_{ik} + \leq e_0^k \text{ for } k = 1, \ldots, n \quad (13)$$

$$x_{ik} \in \{0,1\} \text{ for } (i,k) = (1,1), \ldots, (m,n) \quad (14)$$

The above also can be converted to a MIP with linear constraints using the techniques discussed above.

IV. HEURISTIC FOR TASK MAPPING

A. Greedy Algorithms for Task Mapping

Although the MIP formulation leads to optimal results, solving an MIP can be quite time consuming in practice. Our greedy heuristic for the goal of minimizing the maximum fraction of energy spent at a node (OPT_1) is detailed in Algorithm 1. We first sort the edges in the task graph in non-increasing order of the traffic going on them (step 2). Then, in steps 3 to 14, it tries to map the still unmapped endpoints of each edge (i,j) so as to achieve the minimum increase in the objective function.

Algorithm 1 *GreedyMinMax*: for OPT_1

Input: $D(= DT, DE), N, \mathcal{T}[m][n], \mathcal{R}[n][n][n], f[m],$ $s[m][m], e_o[n]$

Output: $M[m]$: Task Assignment

1: Initialize $M[i] = -1$ for $i = \{1, \ldots, m\}$
2: Sort $(i,j) \in DE$ in non-increasing order of $f[i] \cdot s[i][j]$
3: **for all** (sorted) (i,j) in DE **do**
4:   // Initialize $minmaxCost$ and $minPath$ for this iteration
5:   $minmaxCost = \infty$; $minPath = (-1,-1)$
6:   **for all** (α, β) such that (i,j) can be assigned to them **do**
7:     $M[i] = \alpha$, $M[j] = \beta$ // Temporarily assign (i,j) to $(\alpha \rightarrow \beta)$
8:     $maxCost = maxCost(D, N, \mathcal{T}, \mathcal{R}, f, s, e_0, M)$
9:     **if** $maxCost < minmaxCost$ **then**
10:       // Update $minmaxCost$ and $minPath$
11:       $minmaxCost = maxCost$; $minPath = (\alpha, \beta)$
12:   **if** $minmaxCost > 1$ **then**
13:     **declare failure. stop.** // Checking for feasibility
14:   $M[i] = minPath.\alpha$; $M[j] = minPath.\beta$
15: **return** M

Computational Complexity: Each invocation of *maxCost* takes $\theta(n(m + |DE|))$ time. During Algorithm 1, the sorting takes $O(|DE|\log(|DE|))$ time, and the main loops invokes Algorithm 2 (in step 8) for evaluating the *maxCost* $O(|DE|n^2)$ times. The total time complexity of the algorithm is $O(|DE|(\log(|DE|) + n^3(m + |DE|)))$. Since $|DE| > m$ in a DAG and $|DE| > \log(|DE|)$, this can be simplified to $O(n^3|DE|^2)$.

Algorithm 3 shows our modification to Algorithm 1 for mapping tasks for OPT_2. The algorithm calls *totalCost* subroutine (shown in Algorithm 4) repeatedly (in step 10) to determine the current total cost of the assignment, and chooses the end points of the next edge so as to minimize the total cost. Owing to the similarity in structure, its computational complexity is also $O(n^3|DE|^2)$.

Algorithm 2 *maxCost*: for determining the maximum fraction of energy spent at a node

---

Input: $D(= DT, DE), N, \mathcal{T}[m][n], \mathcal{R}[n][n][n], f[m],$ $s[m][m], e_o[n], M[m]$

Output: *maxCost*: Maximum fraction of energy spent at any node

1: $maxCost = 0$ // Initialize max cost
2: **for all** $k \in N$ **do**
3:   $cost = 0$ // Initialize node cost
4:   **for all** $i \in DT$ **do**
5:     **if** $M[i] == k$ **then**
6:       // Increment computation cost
7:       $cost = cost + f[i] \cdot \mathcal{T}[i][k]$
8:   **for all** $(i,j) \in DE$ **do**
9:     **if** $M[i] \neq 1$ AND $M[j] \neq 1$ **then**
10:       // Increment communication cost
11:       $cost = cost + f[i] \cdot s[i][j] \cdot \mathcal{R}[M[i]][M[j]][k]$
12:   **if** $cost/e_0[k] > maxCost$ **then**
13:     $maxCost = cost/e_0[k]$
14: **return** $maxCost$

Algorithm 3 *GreedyMinTotal*: for OPT_2

Input: $D(= DT, DE), N, \mathcal{T}[m][n], \mathcal{R}[n][n][n], f[m],$ $s[m][m], e_o[n]$

Output: $M[m]$: Task Assignment

1: Initialize $M[i] = -1$ for $i = \{1, \ldots, m\}$
2: Sort $(i,j) \in DE$ in non-increasing order of $f[i] \cdot s[i][j]$
3: **for all** (sorted) (i,j) in DE **do**
4:   // Initialize $mintotalCost$ for this iteration
5:   $mintotalCost = \infty$
6:   $minPath = (-1,-1)$
7:   **for all** (α, β) such that (i,j) can be assigned to them **do**
8:     $M[i] = \alpha$
9:     $M[j] = \beta$ // Temporarily assign (i,j) to $(\alpha \rightarrow \beta)$
10:     $totalCost = totalCost(D, N, \mathcal{T}, \mathcal{R}, f, s, e_0, M)$
11:     **if** $totalCost < mintotalCost$ **then**
12:       // Update mintotalCost
13:       $mintotalCost = totalCost$
14:       $minPath = (\alpha, \beta)$
15:   $maxCost = maxCost(D, N, \mathcal{T}, \mathcal{R}, f, s, e_0, M)$
16:   **if** $maxCost > 1$ **then**
17:     **declare failure. stop.** // Checking for feasibility
18:   $M[i] = minPath.\alpha$
19:   $M[j] = minPath.\beta$
20: **return** M

B. Worst-case Analysis

Since both *GreedyMinMax* and *GreedyMinTotal* are heuristics, we explored the situations when they can give sub-optimal results. We introduce the notion of the *cost of an algorithm* for this purpose – the cost of *GreedyMinMax* is defined a the maximum fraction of energy spent in one round at any node in N, and the cost of *GreedyMinTotal* is the total energy spent by all the nodes in N in one round, when tasks are mapped according to the heuristic.

Algorithm 4 *totalCost*: for determining the total energy spent in the system

Input: $D(= DT, DE), N, \mathcal{T}[m][n], \mathcal{R}[n][n][n], f[m],$
$\quad s[m][m], e_o[n], M[m]$
Output: *totalCost*: Total energy spent by nodes in N
1: $totalCost = 0$ // Initialize total cost
2: **for all** $k \in N$ **do**
3: $\quad cost = 0;$ // Initialize node cost
4: $\quad$ **for all** $i \in DT$ **do**
5: $\quad\quad$ **if** $M[i] == k$ **then**
6: $\quad\quad\quad$ // Increment computation cost
7: $\quad\quad\quad cost = cost + f[i] \cdot \mathcal{T}[i][k]$
8: $\quad$ **for all** $(i, j) \in DE$ **do**
9: $\quad\quad$ **if** $M[i] \neq 1$ AND $M[j] \neq 1$ **then**
10: $\quad\quad\quad$ // Increment communication cost
11: $\quad\quad\quad cost = cost + f[i] \cdot s[i][j] \cdot \mathcal{R}[M[i]][M[j]][k]$
12: $\quad totalCost = totalCost + cost$
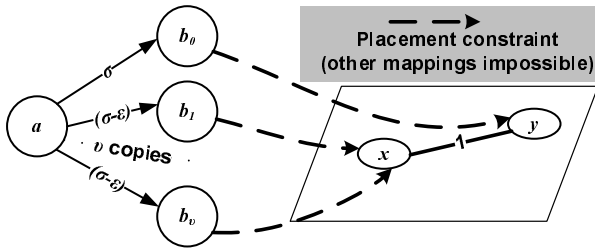13: **return** $totalCost$

Fig. 2. Scenario for worst case performance of *GreedyMinMax* and *GreedyMinTotal*.

Theorem 1. For any integer $\upsilon \geq 1$, there are problem instances for which the cost of *GreedyMinMax* (*GreedyMinTotal*) is arbitrarily close to $\upsilon \times \text{OPT}_1$ ($\upsilon \times \text{OPT}_2$).

Proof. Consider a situation as illustrated in Figure 2. $T_{ax} = T_{ay} = 0$, and the other tasks can only be placed on the nodes indicated by the arrows. Let us also assume that $f_a = 1$, $e_0^x = e_0^y = e_0$, and both nodes in N spend one unit of energy per unit of data transmitted on the link between them. Finally, $e_0 \gg k \gg \varepsilon > 0$. The optimal solution, both for OPT_1 and OPT_2, is to place a on node x, thereby causing only the data on the (a, b_0) edge in DE to go on the network, costing k units of energy to be spent by node x (and the entire system) in each round. The greedy algorithms, however, start with placing the costliest edge (a, b_0) in the best possible manner, co-locating a and b_0 on node y. This leads to $\upsilon \times (k - \varepsilon)$ traffic to go over the $y \to x$ link.

$$\text{We thus get:} \quad \text{OPT}_1 = \frac{1}{e_0}\sigma \quad (15)$$

$$\Rightarrow \text{cost}(GreedyMinMax) = \frac{1}{e_0}\upsilon \times (\sigma - \varepsilon) \approx \upsilon \times \text{OPT}_1 \quad (16)$$

$$\text{Similarly, } \text{OPT}_2 = 2\sigma \quad (17)$$

$$\Rightarrow \text{cost}(GreedyMinTotal) = 2\upsilon \times (\sigma - \varepsilon) \approx \upsilon \times \text{OPT}_2 \quad (18)$$

hence proving the theorem.

Theorem 2. There are problem instances for which *GreedyMinMax* and *GreedyMinTotal* will terminate in failure although a feasible solution exists.

Proof. Consider the situation as illustrated in Figure 2. However, in this case, assume that $e_0 = k \gg \varepsilon > 0$. The optimal solution (given by the MIP formulation) will still place task a on node y, while the Greedy algorithms will try to place it on node y. Note that for $\upsilon \geq 2$, this will lead to an infeasible solution, as the nodes end up spending $> e_0$ energy. The proof follows.

V. TASK MAPPING WITH MULTI-PATH ROUTING

In many WSN applications, multiple routes are possible between a pair of nodes. In this section, we provide generalized versions of our problem formulations to incorporate this condition. The following changes are made to the model:

- We assume that a constant Φ number of paths are possible to be taken between any pair of nodes β and γ in N.
- We further assume that for each pair of communicating tasks (i, j) mapped to nodes β and γ respectively, one of the Φ $\beta \to \gamma$ paths (say ρ) is chosen. Note that for another pair of communicating tasks (s, t) mapped to β and γ, another $\beta \to \gamma$ path ρ' can be chosen.
- To incorporate the above, we redefine the *routing energy cost matrix* $\mathcal{R}$ to be a $n \times n \times n \times \Phi$ matrix, with $\mathcal{R}_{\beta\gamma k\rho}$ denoting the energy consumed per unit of data at node k while routing messages from node β to γ, using the ρth routing option.
- The task-mapping algorithms, apart from determining the task mapping M, also need to provide the **routing path choice mapping** $P : DE \to \{1, 2, \ldots, \Phi\}$.
- The communication cost at node k is now given by:

$$C_{\text{comm}}^k = \sum_{e(=(i,j)) \in DE} f_i \cdot s_{ij} \cdot \mathcal{R}_{M(i)M(j)kP(e)} \quad (19)$$

A. MIP Formulation for OPT_1 when Multi-Path Routing is Possible

The problem of task-mapping and route choice to minimize the maximum fraction of energy spent at a node can thus be formulated as:

Inputs:

- $D = (DT, DE)$: Data-driven Task Graph
- f_i: Firing rate for task i
- s_{ij}: Size of data transferred from task i to j on each invocation of i
- N: Network description
- $\mathcal{T}$: Task execution energy matrix
- $\mathcal{R}$: Routing energy cost matrix, as modified above.

Outputs:

- X: Assignment Matrix. x_{ik} is binary.
- Z: Routing Path Choice Matrix. $z_{e\rho}$ is binary, and is 1 if the traffic over edge e in DE is routed along the ρth path.

Optimization Goal:

$$\text{minimize } c$$

Constraints:

$$\sum_{k=1}^{n} x_{ik} = 1 \text{ for } i = 1, 2, \ldots, m \qquad (20)$$

$$\sum_{\rho=1}^{\Phi} z_{e\rho} = 1 \text{ for each } e \qquad (21)$$

$$\frac{1}{e_k^0}\Big(\sum_{e=(i,j)\in DE} \sum_{\beta=1}^{n}\sum_{\gamma=1}^{n}\sum_{\rho=1}^{C} f_i \cdot s_{ij} \cdot x_{i\beta} \cdot x_{j\gamma} \cdot z_{e\rho} \cdot \mathcal{R}_{\beta\gamma k\rho}\Big)+$$

$$\sum_{i=1}^{m} f_i \cdot \mathcal{T}_{ik} \cdot x_{ik} \leq c \text{ for } k = 1, \ldots, n \qquad (22)$$

$$x_{ik} \in \{0,1\} \text{ for } (i,k) = (1,1), \ldots, (m,n) \qquad (23)$$

$$z_{e\rho} \in \{0,1\} \text{ for each combination of } (e,\rho) \qquad (24)$$

$$0 \leq c \leq 1 \qquad (25)$$

Note that the above is an MIP since c is real whereas x_{ik} and $z_{e\rho}$ are binary integers. Also, it is not a linear program since product terms $x_{i\beta} \cdot x_{j\gamma} \cdot z_{e\rho}$ appear in the constraints.

The above problem can be converted to a linear MIP by repeatedly applying the techniques discussed in the previous section. We first absorb each $(x_{i\beta}, x_{j\gamma})$ pair into a variable $y_{i\beta j\gamma}$, and then introduce another set of variables, one to absorb each $(y_{i\beta j\gamma}, z_{e\rho})$ pair, to get the following:

$$y_{i\beta j\gamma} - x_{i\beta} \leq 0 \qquad (26)$$

$$y_{i\beta j\gamma} - x_{j\gamma} \leq 0 \qquad (27)$$

$$x_{i\beta} + x_{j\gamma} - y_{i\beta j\gamma} \leq 1 \qquad (28)$$

$$u_{i\beta j\gamma\rho} - y_{i\beta j\gamma} \leq 0 \qquad (29)$$

$$u_{i\beta j\gamma\rho} - z_{e\rho} \leq 0 \qquad (30)$$

$$y_{i\beta j\gamma} + z_{e\rho} - u_{i\beta j\gamma\rho} \leq 1 \qquad (31)$$

B. MIP Formulation for OPT_2 when Multi-Path Routing is Possible

Using our formulation, the objective of solving the problem to minimize the *total energy* spent by the system can be formulated as follows:

Inputs:
- $D = (DT, DE)$: Data-driven Task Graph
- f_i: Firing rate for task i
- s_{ij}: Size of data transferred from task i to j on each invocation of i
- N: Network description
- $\mathcal{T}$: Task execution energy matrix
- $\mathcal{R}$: Routing energy cost matrix, as modified above.

Outputs:
- X: Assignment Matrix. x_{ik} is binary.

- Z: Routing Path Choice Matrix. $z_{e\rho}$ is binary.

Optimization Goal:

$$\text{minimize} \sum_{k=1}^{n} \left(\begin{array}{l} \sum_{i=1}^{m} f_i \cdot \mathcal{T}_{ik} \cdot x_{ik}+ \\ \sum_{e=(i,j)\in DE} \sum_{\beta=1}^{n}\sum_{\gamma=1}^{n}\sum_{\rho=1}^{\Phi} \left(\begin{array}{l} f_i \cdot s_{ij} \cdot x_{i\beta}\cdot \\ x_{j\gamma} \cdot z_{e\rho} \cdot \mathcal{R}_{\beta\gamma k\rho} \end{array} \right) \end{array} \right)$$

Constraints:

$$\sum_{k=1}^{n} x_{ik} = 1 \text{ for } i = 1, 2, \ldots, m \qquad (32)$$

$$\sum_{\rho=1}^{\Phi} z_{e\rho} = 1 \text{ for each combination of } (e, \beta, \gamma) \qquad (33)$$

$$\sum_{e=(i,j)\in DE} \sum_{\beta=1}^{n}\sum_{\gamma=1}^{n}\sum_{\rho=1}^{\Phi} f_i \cdot s_{ij} \cdot x_{i\beta} \cdot x_{j\gamma} \cdot z_{e\rho} \cdot \mathcal{R}_{\beta\gamma k\rho}+$$

$$\sum_{i=1}^{m} f_i \cdot \mathcal{T}_{ik} \cdot x_{ik} \leq e_0^k \text{ for } k = 1, \ldots, n$$

$$\qquad (34)$$

$$x_{ik} \in \{0,1\} \text{ for } (i,k) = (1,1), \ldots, (m,n) \qquad (35)$$

$$z_{e\rho} \in \{0,1\} \text{ for each combination of } e, \beta, \gamma, \rho \qquad (36)$$

Note that the above can be converted to a MIP with linear constraints using the linearization techniques used by us.

C. Greedy Heuristics for Task-mapping with Multi-Path Routing

In view of the changed system model, we can modify the algorithms proposed in Section IV. Our greedy heuristic for the goal of minimizing the maximum fraction of energy spent at a node (OPT_1) is detailed in Algorithm 5. First, the algorithm sorts the edges in the task graph in non-increasing order of the traffic going on them (step 3). Then, for each still-unmapped endpoints of each edge (i, j), it iterates over all possible pair of nodes to assign the end points to, and all possible paths between them (steps 4-23).

and determine the best route to be taken by the data items transferred between i and j them, so as to achieve the minimum increase in the objective function.

Computational Complexity: Each invocation of *maxCostM* takes $\theta(n(m + |DE|))$ time. During Algorithm 5, the sorting takes $O(|DE|\log(|DE|))$ time, and the main loops invokes Algorithm 6 for evaluating the *maxCost* $O(|DE|n^2\Phi)$ times. The total time complexity of the algorithm is $O(|DE|(\log(|DE|) + n^3(m + |DE|)\Phi))$. Since $|DE| > m$ in a DAG and $|DE| > \log(|DE|)$, this can be simplified to $O(n^3|DE|^2\Phi)$.

Algorithm 7 shows our modification to Algorithm 5 for mapping tasks for OPT_2. The algorithm calls *totalCostM* subroutine (shown in Algorithm 8) repeatedly to determine the current total cost of the assignment, and chooses the end points of the next edge and the path choice so as to minimize the total

---

Algorithm 5 *GreedyMinMaxM*: for OPT_1 with Multi-Path Routing

---

Input: $D(= DT, DE), N, \mathcal{T}[m][n], \mathcal{R}[n][n][n][\Phi], f[m],$
    $s[m][m], e_o[n]$

Output: $M[m]$: Task Assignment, $P[|DE|]$: Routing Path
    Choice

 1: Initialize $M[i] = -1$ for $i = \{1, \dots, m\}$
 2: Initialize all entries $P[e] = -1$
 3: Sort $(i, j) \in DE$ in non-increasing order of $f[i] \cdot s[i][j]$
 4: **for all** (sorted) $e = (i, j)$ in DE **do**
 5:     // Initialize minmaxCost for this iteration
 6:     $minmaxCost = \infty$
 7:     $minPath = (-1, -1)$
 8:     **for all** (α, β) such that (i, j) can be assigned to them **do**
 9:         $M[i] = \alpha$
10:         $M[j] = \beta$ // Temporarily assign (i, j) to $(\alpha \to \beta)$
11:         **for** $\rho = 1$ to Φ **do**
12:             // Temporarily choose the ρth routing option
13:             $P[e] = \rho$
14:             $maxCost =$
                    $maxCostM(D, N, \mathcal{T}, \mathcal{R}, f, s, e_0, M, P)$
15:             **if** $maxCost < minmaxCost$ **then**
16:                 // Update mintotalCost
17:                 $minmaxCost = maxCost$
18:                 $minPath = (\alpha, \beta, \rho)$
19:     **if** $minmaxCost > 1$ **then**
20:         **declare failure. stop.** // Checking for feasibility
21:     $M[i] = minPath.\alpha$
22:     $M[j] = minPath.\beta$
23:     $P[e][M[i]][M[j]] = minPath.\rho$
24: **return** M, P

Algorithm 6 *maxCostM*: for determining the maximum fraction of energy spent at a node

---

Input: $D(= DT, DE), N, \mathcal{T}[m][n], \mathcal{R}[n][n][n], f[m],$
    $s[m][m], e_o[n], M[m], P[|DE|][n][n]$

Output: *maxCostM*: Maximum fraction of energy spent at
    any node

 1: $maxCostM = 0$ // Initialize max cost
 2: **for all** $k \in N$ **do**
 3:     $cost = 0$ // Initialize node cost
 4:     **for all** $i \in DT$ **do**
 5:         **if** $M[i] == k$ **then**
 6:             // Increment computation cost
 7:             $cost = cost + f[i] \cdot \mathcal{T}[i][k]$
 8:     **for all** $e = (i, j) \in DE$ **do**
 9:         **if** $M[i] \neq -1$ AND $M[j] \neq -1$ AND $P[e] \neq -1$ **then**
10:             // Increment communication cost
11:             $cost = cost + f[i] \cdot s[i][j] \cdot \mathcal{R}[M[i]][M[j]][k][P[e]]$
12:     **if** $cost/e_0[k] > maxCostM$ **then**
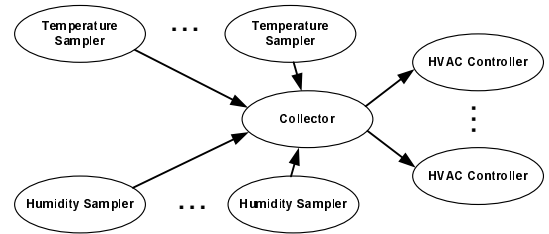13: $maxCostM = cost/e_0[k]$
14: **return** $maxCostM$

Fig. 3. A task graph for HVAC management.

cost. Owing to the similarity in structure, its computational complexity is also $O(n^3 |DE|^2 \Phi)$.

Worst Case Analysis: Since *GreedyMinMaxM* and *GreedyMinTotalM* are generalized versions of the algorithms discussed in Section IV, the problem instance discussed in Section IV-B acts as a special case of the task-mapping problem with multi-path routing, with the maximum number of routes $\Phi = 1$. Therefore, the same worst-case bounds hold for the algorithms discussed in this section also.

VI. Evaluation

As stated in Section II-E, the proper manner to evaluate the performance of task-mapping techniques for networked sensing applications is to use task graphs derived from real applications, as opposed to randomly generated ones. In this section, we describe two real-world networked sensing application scenarios, and the results obtained by our techniques on them.

A. Reference Applications

For evaluating our techniques, we use two real-world applications in this paper. The first is a building environment management application for monitoring heating, ventilation and air-conditioning (HVAC), similar in spirit to other applications in the literature [3]. We consider a set of nodes spread across a building, with each node possibly attached to a temperature sensor, a humidity sensor and an actuator that can control the temperature and humidity of a region. The aim of the system is to maintain desirable temperature and humidity levels in each room of the building, by correlating the information from the sensor installed in the room, and using it to drive actuation.

Figure 3 describes our application as a data-driven task graph. The *Temperature Sampler* and *Humidity Sampler* tasks – instantiated on the nodes with relevant sensors – sample their surroundings and generate temperature and humidity readings. This data is then sent to the *Collector* task, one of which is placed in each room. Upon processing the data, the *Collector* produces a command for the actuating tasks and sends the data to the *HVAC Controller* task, which is placed on all nodes with an HVAC Actuator and responds to the *Action* data item by adjusting the temperature/humidity controls.

The second application, illustrated in Figure 4, describes a highway traffic management system, similar in spirit to [2]. In this case, two different sub-goals must be achieved - regulating the speed of vehicles on the highway by controlling speed limit displays, and controlling the access to the highway by means of red/green signals on the ramps. The highway is divided into sectors, and sensors are deployed on the

Algorithm 7 *GreedyMinTotalM*: for OPT_2 with Multi-Path Routing

---

Input: $D(= DT, DE), N, \mathcal{T}[m][n], \mathcal{R}[n][n][n], f[m],$
$\quad s[m][m], e_o[n]$

Output: $M[m]$: Task Assignment, $P[|DE|]$: Routing Path Choice

1: Initialize $M[i] = -1$ for $i = \{1, \ldots, m\}$
2: Initialize all entries $P[e][\beta][\gamma] = -1$
3: Sort $(i, j) \in DE$ in non-increasing order of $f[i] \cdot s[i][j]$
4: **for all** (sorted) $e = (i, j)$ in DE **do**
5: $\quad$ // Initialize mintotalCost for this iteration
6: $\quad mintotalCost = \infty$
7: $\quad minPath = (-1, -1)$
8: $\quad$ **for all** (α, β) such that (i, j) can be assigned to them **do**
9: $\quad\quad M[i] = \alpha$
10: $\quad\quad M[j] = \beta$ // Temporarily assign (i, j) to $(\alpha \rightarrow \beta)$
11: $\quad\quad$ **for** $\rho = 1$ to Φ **do**
12: $\quad\quad\quad$ // Temporarily choose the ρth routing option
13: $\quad\quad\quad P[e] = \rho$
14: $\quad\quad\quad totalCost =$
$\quad\quad\quad\quad\quad totalCostM(D, N, \mathcal{T}, \mathcal{R}, f, s, e_0, M, P)$
15: $\quad\quad\quad$ **if** $totalCost < mintotalCost$ **then**
16: $\quad\quad\quad\quad$ // Update mintotalCost
17: $\quad\quad\quad\quad mintotalCost = totalCost$
18: $\quad\quad\quad\quad minPath = (\alpha, \beta, \rho)$
19: $\quad maxCost = maxCostM(D, N, \mathcal{T}, \mathcal{R}, f, s, e_0, M, P)$
20: $\quad$ **if** $maxCost > 1$ **then**
21: $\quad\quad$ **declare failure. stop.** // Checking for feasibility
22: $\quad M[i] = minPath.\alpha$
23: $\quad M[j] = minPath.\beta$
24: $\quad P[e] = minPath.\rho$
25: **return** M, P

Algorithm 8 *totalCostM*: for determining the total energy spent in the system

---

Input: $D(= DT, DE), N, \mathcal{T}[m][n], \mathcal{R}[n][n][n], f[m],$
$\quad s[m][m], e_o[n], M[m], P[|DE|]$

Output: *totalCostM*: Total energy spent by nodes in N

1: $totalCostM = 0$ // Initialize total cost
2: **for all** $k \in N$ **do**
3: $\quad cost = 0;$ // Initialize node cost
4: $\quad$ **for all** $i \in DT$ **do**
5: $\quad\quad$ **if** $M[i] == k$ **then**
6: $\quad\quad\quad$ // Increment computation cost
7: $\quad\quad\quad cost = cost + f[i] \cdot \mathcal{T}[i][k]$
8: $\quad$ **for all** $e = (i, j) \in DE$ **do**
9: $\quad\quad$ **if** $M[i] \neq -1$ AND $M[j] \neq -1$ AND $P[e] \neq -1$ **then**
10: $\quad\quad\quad$ // Increment comm. cost
11: $\quad\quad\quad cost = cost + f[i] \cdot s[i][j] \cdot \mathcal{R}[M[i]][M[j]][k][P[e]]$
12: $\quad totalCostM = totalCostM + cost$
13: **return** $totalCostM$

---

actuators.

B. Experiments

For evaluating the relative performance of our heuristics, we applied them on the reference applications discussed in Section VI-A, by using our algorithms to map their tasks onto a various simulated target deployments (shown in Figure 5) to map the tasks onto. For the HVAC application, we placed an equal number of temperature and humidity sensors in a grid in a room, and assigned the location of the HVAC actuators randomly. We also placed extra nodes in the room for maintaining connectivity. For the traffic application, we placed forwarding nodes uniformly apart at the edge of the highway, and randomly distributed the speed sensors on the four lanes so that each of them was in range of at least another speed sensor or a forwarding node. Similarly, the presence sensors were randomly distributed on the ramp so that each of them was in range of at least one speed sensor or another presence sensor. The node controlling the ramp signals and the speed limit displays were placed between different sectors, on opposite sides of the road. Note that for both the applications, owing to the placement constraints of the applications, the number of tasks m is $O(n)$ for our experiments, where n is the number of nodes.

Experimental Results: In our experiments, we assumed that all nodes started with a sufficiently high initial energy level e_0. The routing energy cost matrix $\mathcal{R}$ was obtained by using a shortest path algorithm on the network, assuming equal energy spent by all nodes on a route, and all data items were assumed to be of unit size ($s_{ij} = 1$). The task execution energy matrix $\mathcal{T}$ was set up to represent placement constraints: $\mathcal{T}_{ik} = 0$ when task i could be placed on node k, ∞ when it could not. The tasks which performed sensing and actuating were tied to a node with the relevant capabilities. Finally, the f_i for each task was computed as follows: For sensing tasks, f_i was set to 10, and for all other tasks j, f_j was set to the sum of the firing
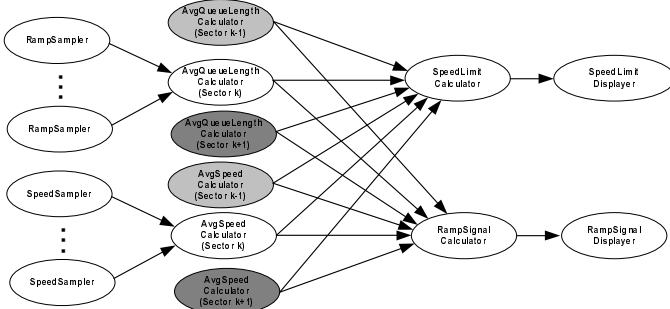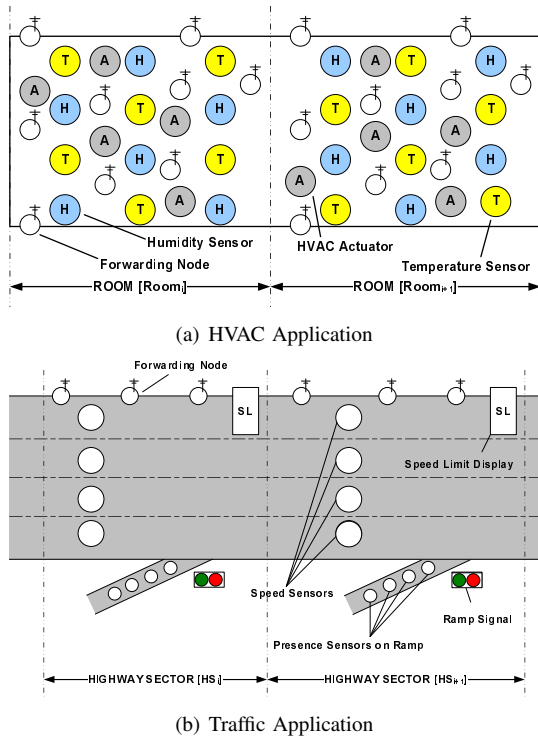

Fig. 4.   An task graph for highway traffic management.

highway lanes and ramps to sense the speed and presence of vehicles, respectively. The sensed data goes through a multi-stage process where it is first aggregated w.r.t. a single sector to derive an average measure (*AvgSpeedCalculator* and *AvgQueueLengthCalculator* tasks). The *SpeedLimitCalculator* and *RampSignalCalculator* tasks take the average speeds and queue lengths produced in the neighboring highway sectors (as shown in the figure), and compute the desired actions to be sent to the *SpeedLimitDisplayer* and *RampSignalDisplayer* tasks, which are located on the nodes attached to the corresponding

(a) HVAC Application

(b) Traffic Application

Fig. 5.   Node placement in reference applications.

(a) HVAC Management

(b) Traffic Management

Fig. 6.   Time taken to compute task-mapping for minimizing maximum energy spent by any node.

rates of tasks on the other ends of the incoming edges. This represented the fact that task j fires whenever there is data available for it. For the multi-path scenario, we generated the routing matrix using the generalized Floyd algorithm [7] with $\Phi = 3$.

We ran our experiments on a PC with a dual-core Pentium processor running at 1.6GHz, with 2GB of RAM. We implemented our greedy algorithm in Java, and solved the MIPS using the lp_solve [8] linear programming toolkit. The time taken for computing task placements for both the applications so as to minimize the maximum fraction of energy spent by any node (OPT_1) is shown in Figure 6. The time taken by the two techniques for placing tasks so as to minimize the total energy spent in the system (OPT_2) is shown in Figure 7.

In our experiments, the Greedy algorithms obtained sub-optimal results only while computing task-mappings for minimizing total energy in some of the HVAC application. For the traffic application, and the for *all* instances of OPT_1 (which we believe is a better indicator of system lifetime), the solution given by the greedy algorithm was the same as the one given by the MIP. Our experiments clearly show that the greedy algorithms take much less time that the MIP formulation in finding the mappings. This showcases the efficacy of the algorithms in solving the task-mapping problem for complex real-world WSN applications.

In experiments conducted where multiple paths were possible (shown in Figures 8 and 9), we see that the time taken by the heuristics still outperforms the MIP solver in terms of time. Note that the graphs for the traffic management application denote the fact that the MIP solver did not terminate in a long time for some instances. As before, the quality of solution given by our heuristics were found to be as good as that of
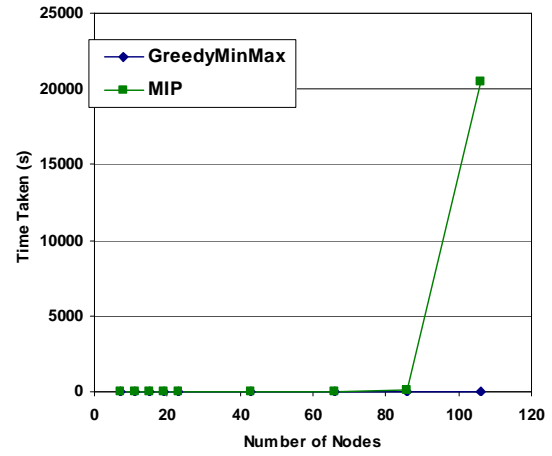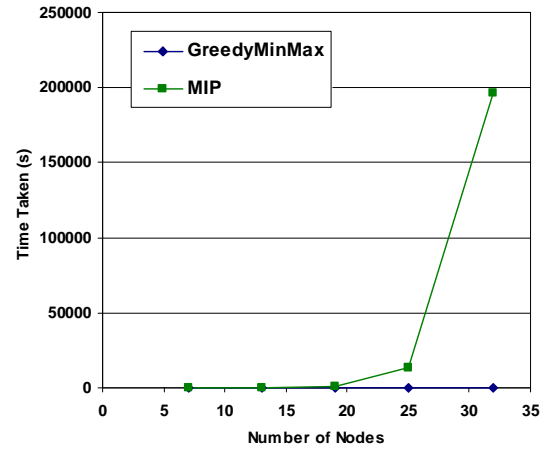
the MIP, with the cost of the (few) outliers not being more than 1.5 times the optimal cost.

In summary, our experimental results support our analysis that the MIP solver will take much more time to provide the (optimal) solution to the task-mapping problems than our heuristics. It is also noteworthy that the solver takes less time to map the tasks in the (simpler) HVAC task graph (Figure 5(a)) as compared to the more complex Traffic application task graph (Figure 5(b)).

VII. RELATED WORK

A large body of work exists, both in the parallel and distributed computing as well as the wireless sensor networking domain, on the problem of mapping tasks of an application onto the nodes of a target system. In this section, we present some closely related work from various domains.

Parallel and Distributed Computing: The task mapping problem [9] is a well studied problem is parallel and distributed computing. In [10], the authors have covered a wide range of mapping problems in such systems and approached to solve them. However, they are mostly concerned with optimizing for *latency*, i.e., minimizing the computation and communication time. In addition, the tasks do not have
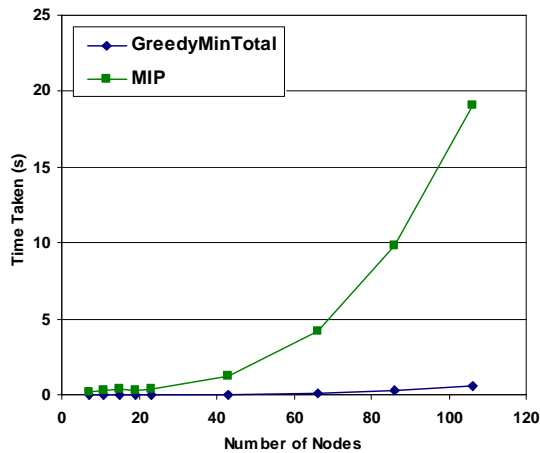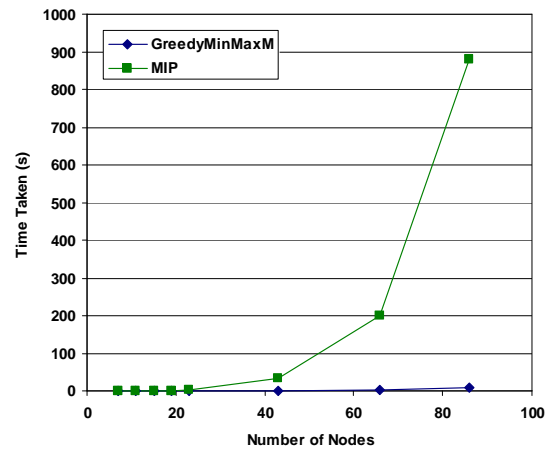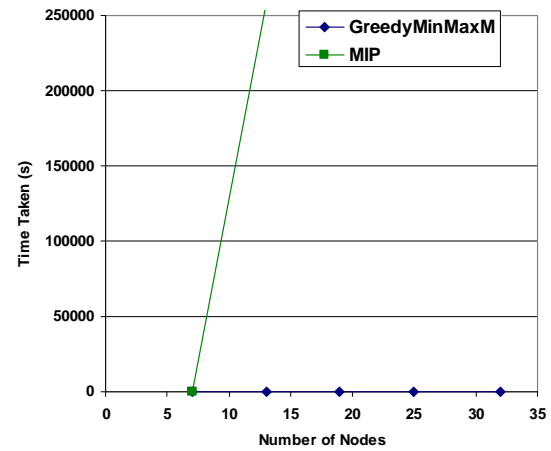
(a) HVAC Management
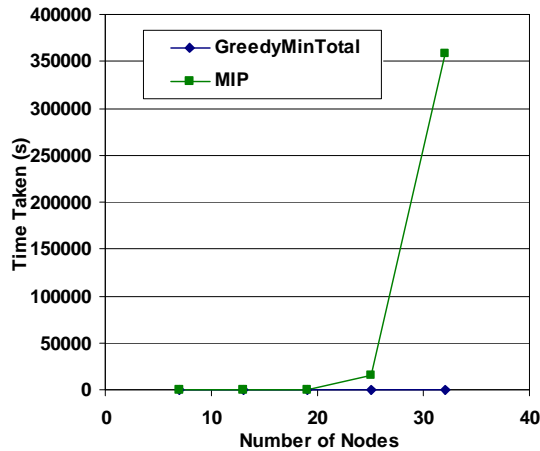
(a) HVAC Management - Multipath

(b) Traffic Management

(b) Traffic Management - Multipath

Fig. 7.   Time taken to compute task-mapping for minimizing total energy spent in the system.

Fig. 8.   Time taken to compute task-mapping for minimizing maximum energy spent by any node (MultiPath).

placement constraints. In [11], the authors include placement constraints in their problem statement using a *task preference matrix*. However, they assume that communication costs are paid only by the end-points, and their optimization goal is the total cost that the system endures for the application.

Heterogeneous Systems: In [12], the authors present a genetic algorithm for placing tasks onto a parallel processor. They also provide an extension for the case where not all tasks can be run on all nodes, by way of assigning each node to a class, and associating a class number with each task. Their algorithm is designed to work for a range of metrics, and they focus on the *minimize total execution time* metric in the paper. However, unlike our work, they assume full control over the message routing. In [13], the authors present algorithms based on the best-first A* technique from artificial intelligence for optimal task placement on heterogeneous systems. The placement constraint is specified as a *placement cost metric* for mapping a task to a particular node. Subject to these costs, the nodes are assumed to be capable of executing any task in the application. However, unlike our work, their optimization goal is to minimize the turnaround time. Also, they assume a dedicated interconnection network, and there are no routing overheads for intermediate nodes. Similarly, recent work such

as that in [14] focus on scheduling jobs on grids by a Multi-Resource Scheduling (MRS) algorithm using virtual maps and resource potentials. However, they also assume a completely connected network, and no routing costs.

Wireless Sensor Networks: A wide variety of work exists in sensor networks to maximize lifetime by reducing the energy spent, mostly using distributed algorithms for sleep-wake scheduling [15]. The work in [16] achieves energy-balance during data-propagation by deciding in each step whether to propagate data one-hop towards the final destination (the sink), or to send data directly to the sink. This randomized choice ensures that the average per sensor energy dissipation is the nearly the same for all sensors in the network. Task placement on sensor networks has also been addressed recently. One of the early works on this topic is [17], where the authors propose an energy-balanced task allocation for collaborative processing in WSNs. However, unlike our work, they focus on single-hop networks only. Further, our system model is more general than theirs in some respects, since they only consider the case where two tasks cannot share the same node. In [18], the authors have provided task placement approaches for unconstrained task graphs with optimization goals such as minimizing total energy. In addition, they also provide the routes taken by mes-

(a) HVAC Management - MultiPath

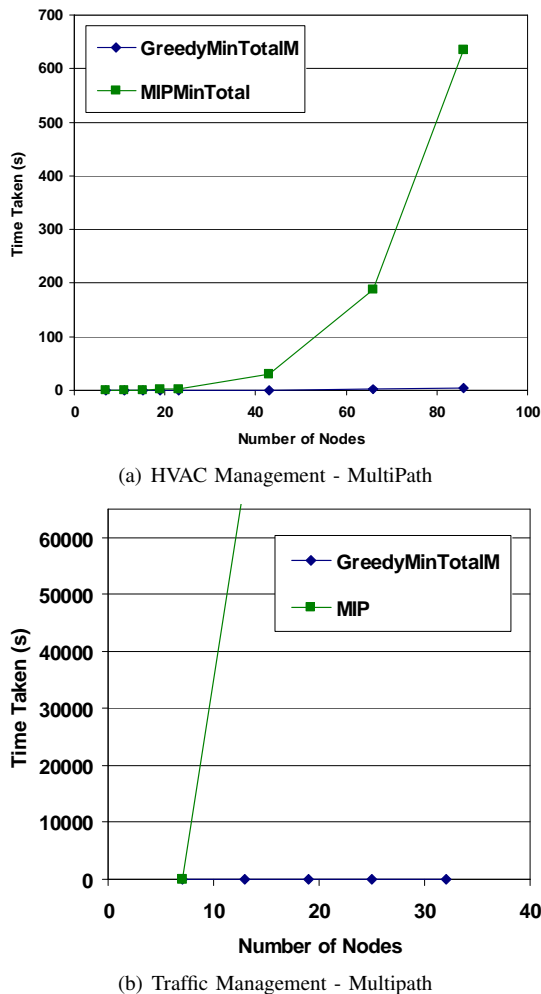(b) Traffic Management - Multipath

Fig. 9.   Time taken to compute task-mapping for minimizing total energy spent in the system (MultiPath).

sages. Finally, efforts such as [19] approach the task-mapping problem for WSNs from a protocol-centric point of view, whereas we take a high-level perspective of the problem to determine a good initial task mapping. Most recently, the work in [20] proposed a scheduling algorithm for achieving energy-balance while assigning tasks with precedence constraints on a heterogenous sensor network. However, unlike our work, they focus only on a single-hop network.

WSN Macroprogramming: With the advent of macroprogramming, several approaches have addressed this problem as it arises due to the high-level of applications divided into tasks. [21] proposes a greedy solution to the *service placement problem*, which is applicable to our context of compiling macroprograms. Similar to our case, their application also has task placement constraints, where certain tasks can be placed only on certain nodes. However, they focus only on task graphs that are *trees*, and not general graphs. Further, their algorithm's goal is to minimize the *total* energy of the system, and does not guarantee that a single node will not be over-penalized. The work in [22] solves the generic role assignment problem, where task placements are specified using *roles*. Their algorithm allows ILP solutions of role assignment onto the nodes of the target system, based on a global optimization

criteria represented in terms of the number of nodes with a particular role. Unlike their case, our heuristics are meant for solving an offline version of the problem, and the optimization goals more tied to the energy-consumption at the nodes.

In the recent past, several macroprogramming languages have been proposed, covering a wide range of programming styles. Kairos [23] (and later, Pleiades [24]) provide an imperative approach to task the nodes in the WSN. However, they address the problem of distribution actions onto nodes by dividing the program's control flow graph (CFG) into *nodecuts*, and then placing them on individual nodes. Their heuristic aims to minimize the total number of edges in a program's CFG that cross from one nodecut to another. However, unlike our work, it is not clear what underlying routing costs they are assume, and how they address placement constraints. Regiment [25] is a macroprogramming system using which application developers can specify their programs in a Haskell-like functional programming manner. However, since their underlying routing mechanism uses only spanning-trees, the mapping problem they solve during compilation is different from ours. In their work on COSMOS [26], the authors have presented the *mPL* macroprogramming language and the *mOS* operating system which can be used to program WSNs by way of task graphs. This is similar to our work, and indeed, our system abstraction and techniques can be used during the compilation process in COSMOS also. However, Figure 1 in [26] seems to suggest that they currently address only the case where *all* nodes of a single type have the same set of tasks running on them. Finally, MacroLab [27] provides a Matlab-like interface to WSN application developers, so that they can use operations such as `addition`, `max`, and `find` on sensor data addressed as *macrovectors*. Since this paradigm focuses on accessing and operating on data presented in matrix form, sometimes using different implementations (centralized versus distributed) of the same operation (e.g. `max`), the task-mapping problem they address is different from the one discussed in our work.

To summarize, although task-mapping as a general problem has been studied for a long time, its application in sensor network macroprogramming is relatively new, and brings in several new aspects. To the best of our knowledge, no existing work addresses the problem of mapping task graphs with placement constraints on arbitrary heterogenous networks with known routing costs so as to achieve energy balance.

VIII. CONCLUDING REMARKS

In this paper, we formalized the problem of mapping tasks with placement-constraints and data-rates as it arises in the context of designing applications for wireless sensor networks using data-driven macroprogramming. These applications can process data streams or items in-network to take decisions about actuation. We provided mathematical formulations for two energy-related optimization goals – minimizing the maximum fraction of energy consumed in a node and minimizing the total energy consumed in the sensor network. For each of these goals, we focused both on the case when a single route was available between any two nodes, as well as when

multiple routes were possible. We used our modeling framework to provide mathematical formulations to solve these four problem instances, and demonstrated linearization techniques to convert them into mixed-integer programs (MIP). We also provided greedy heuristics for the above problem scenarios, and provided worst-case performance bounds for the same. In spite of the worst-case performance possible for specially crafted problem instances, our heuristics were shown to outperform the MIP formulation by several orders of magnitudes of time for real-world WSN applications, while not severely compromising in the quality of the solutions.

The area of mapping data-driven task graphs on sensor networks is still developing. Our hope is that the concise model described in this paper will aid future research in this area, and the MIP formulations can be used to compute the optimal placements where time of computation is not an issue. Further, we acknowledge that later in the life of the WSN applications, distributed protocols will be needed to re-assign the tasks in view of changing operating circumstances. However, our techniques (and other technique based on our models) will provide good initial task placements. Our immediate future work is to reduce the complexity of the heuristics, as well as to explore better polynomial time approximation algorithms. Additionally, we are working on integrating our algorithms into the compiler [28] of a pre-existing data-driven macroprogramming framework.

ACKNOWLEDGMENT

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 *Workshop on End-to-end Sense-and-respond Systems (EESR)*, 2005.

[2] T. T. Hsieh, "Using sensor networks for highway and traffic applications," *IEEE Potentials*, vol. 23, no. 2, 2004.

[3] M. Dermibas, "Wireless sensor networks for monitoring of large public buildings," University at Buffalo, Tech. Rep., 2005.

[4] T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J. A. Stankovic, T. F. Abdelzaher, J. Hui, and B. Krogh, "Vigilnet: An integrated sensor network system for energy-efficient surveillance," *ACM Trans. Sen. Netw.*, vol. 2, no. 1, pp. 1–38, 2006.

[5] A. Pathak and V. K. Prasanna, "Energy-efficient task mapping for data-driven sensor network macroprogramming," in *International Conference on Distributed Computing in Sensor Systems (DCOSS)*, June 2008.

[6] G. Nemhauser, A. RinnooyKan, and M. Todd, Eds., *Optimiztions: Handbooks in Operations Research and Management Science*. North-Holland, 1989, vol. 1.

[7] J. R. Evans and E. Minieka, *Optimization Algorithms for Networks and Graphs, Second Edition*. CRC Press, 2nd Edition, 1992.

[8] "LP Solve," http://lpsolve.sourceforge.net/.

[9] S. H. Bokhari, "On the mapping problem," *IEEE Transactions on Computers*, vol. C-30, no. 3, pp. 207–214, March 1981.

[10] H. El-Rewini, T. G. Lewis, and H. H. Ali, *Task scheduling in parallel and distributed systems*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1994.

[11] P.-Y. R. Ma, E. Lee, and M. Tsuchiya, "A task allocation model for distributed computing systems," *IEEE Transactions on Computers*, vol. C-31, no. 1, pp. 41–47, January 1982.

[12] C. Ravikumar and A. Gupta, "Genetic algorithm for mapping tasks onto a reconfigurable parallel processor," *IEE Proceedings on Computers and Digital Techniques*, vol. 142, no. 2, pp. 81–86, March 1995.

[13] M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous computing systems," in *Sixth Heterogeneous Computing Workshop (HCW '97)*, April 1997.

[14] B. K. B. Tat, B. Veeravalli, T. Hung, and S. S. C. Wee, "A co-ordinate based resource allocation strategy for grid environments," in 6th *IEEE International Symposium on Cluster Computing and Grid (CCGRID)*, May 2006, pp. 561–567.

[15] S. K. Prasad and A. Dhawan, "Distributed algorithms for lifetime of wireless sensor networks based on dependency structure among cover sets," in *Intl High Performance Computing (HiPC)*, 2007.

[16] C. Efthymiou, S. Nikoletseas, and J. Rolim, "Energy balanced data propagation in wireless sensor networks," *Wireless Networks (WINET) Journal, Special Issue on "Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks*, 2006.

[17] Y. Yu and V. K. Prasanna, "Energy-balanced task allocation for collaborative processing in wireless sensor networks," *MONET*, vol. 10, no. 1-2, pp. 115–131, 2005.

[18] Y. Tian, E. Ekici, and F. Ozguner, "Energy-constrained task mapping and scheduling in wireless sensor networks," in *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference RPMSN Workshop*, 2005.

[19] K. H. Low, W. Leow, and J. M.H. Ang, "Autonomic mobile sensor network with self-coordinated task allocation and execution," *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, vol. 36, no. 3, pp. 315–327, May 2006.

[20] L. K. Goh and B. Veeravalli, "An energy-balanced task scheduling heuristic for heterogeneous wireless sensor networks," in *International Conference on High Performance Computing (HiPC)*, 2008, pp. 257–268.

[21] Z. Abrams and J. Liu, "Greedy is good: On service tree placement for in-network stream processing," in *ICDCS '06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*. Washington, DC, USA: IEEE Computer Society, 2006, p. 72.

[22] C. Frank and K. Römer, "Solving generic role assignment exactly," in *IPDPS*, 2006.

[23] R. Gummadi, O. Gnawali, and R. Govindan, "Macro-programming wireless sensor networks using kairos," in *International Conference on Distributed Computing in Sensor Systems (DCOSS)*. Springer, 2005, pp. 126–140.

[24] N. Kothari, R. Gummadi, T. Millstein, and R. Govindan, "Reliable and efficient programming abstractions for wireless sensor networks," in *PLDI '07: Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation*. New York, NY, USA: ACM, 2007, pp. 200–210.

[25] R. Newton, G. Morrisett, and M. Welsh, "The regiment macroprogramming system," in *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*. New York, NY, USA: ACM, 2007, pp. 489–498.

[26] A. Awan, S. Jagannathan, and A. Grama, "Macroprogramming heterogeneous sensor networks using cosmos," in *EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*. New York, NY, USA: ACM, 2007, pp. 159–172.

[27] T. W. Hnat, T. I. Sookoor, P. Hooimeijer, W. Weimer, and K. Whitehouse, "Macrolab: a vector-based macroprogramming framework for cyber-physical systems," in *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*. New York, NY, USA: ACM, 2008, pp. 225–238.

[28] A. Pathak, L. Mottola, A. Bakshi, G. P. Picco, and V. K. Prasanna, "A compilation framework for macroprogramming networked sensors," in *Proc. of the the 3rd Int. Conf. on Distributed Computing on Sensor Systems (DCOSS)*, 2007.

Animesh Pathak received the B.Tech degree in Computer Science and Engineering from the Institute of Technology, Banaras Hindu University, Varanasi, India in 2003, and a PhD in Computer Engineering from the University of Southern California in 2008. He is currently a researcher with the ARLES project-team at INRIA Paris-Rocquencourt. His research interests include parallel and distributed systems including networked sensor systems, and pervasive computing. He has published and presented his work at several international workshops and conferences. He is a member of the IEEE.

Viktor K. Prasanna is Charles Lee Powell Chair in Engineering in the Ming Hsieh Department of Electrical Engineering and Professor of Computer Science at the University of Southern California.

He is the executive director of the USC-Infosys Center for Advanced Software Technologies (CAST). He is also an associate member of the Center for Applied Mathematical Sciences (CAMS) at USC, and a member of the USC-Chevron Center of Excellence for Research and Academic Training on Interactive Smart Oilfield Technologies (Cisoft). His research interests include parallel and distributed systems including networked sensor systems, embedded systems, configurable architectures and high performance computing.

He is the Steering Co-chair of the International Parallel and Distributed Processing Symposium and is the Steering Chair of the International Conference on High Performance Computing (HiPC). He has served on the editorial boards of the Journal of Parallel and Distributed Computing, Proceedings of the IEEE, IEEE Transactions on VLSI Systems, and IEEE Transactions on Parallel and Distributed Systems. He served as the Editor-in-Chief of the IEEE Transactions on Computers during 2003-06. He was the founding Chair of the IEEE Computer Society Technical Committee on Parallel Processing. He is a Fellow of the IEEE and the ACM. He is a recipient of the 2005 Okawa Foundation Grant and 2009 Outstanding Engineering Alumnus Award from the Pennsylvania State University. His website is at http://ceng.usc.edu/~prasanna