Energy Management in Wireless Mobile Systems Using Dynamic Task Assignment

Priti Aghera[1], Jinseok Yang[1], Piero Zappi[2], Dilip Krishnaswamy[2], Ayse Coskun[3], and Tajana Simunic Rosing[1]*

[1]University of California San Diego, San Diego, California, 92093, USA {paghera, jiy011, tajana}@ucsd.edu

[2]Qualcomm Research Center, San Diego, California, 92121, USA, {dilipk, pzappi}@qti.qualcomm.com

[3]Boston University, Boston, Massachusetts, 02215, acoskun@bu.edu

* Corresponding author: Tajana S. Rosing

Address:
University of California San Diego
Department of Computer Science and Engineering
9500 Gilman Dr
San Diego, California, 92093, USA

Office	: +1 (858) 534-4868
Fax	: (858) 534-7029
Email	: tajana@ucsd.edu

Date of Receiving: **to be completed by the Editor**

Date of Acceptance: **to be completed by the Editor**

Energy Management in Wireless Mobile Systems Using Dynamic Task Assignment

Priti Aghera, Jinseok Yang, Piero Zappi, Dilip Krishnaswamy, Ayse Coskun, and Tajana S. Rosing

Abstract *— Wireless mobile sensing systems are hierarchical and heterogeneous in nature with components that have different energy and performance capabilities. In such systems allocation of tasks to different devices affects both performance and the entire system battery lifetime. In this paper we formulate the problem of optimal task assignment with objectives related to minimizing the total system energy consumption and maximizing the system lifetime as an Integer Linear Program (ILP). We describe a heuristic algorithm and two dynamic graph-based partitioning algorithms that are computationally efficient and that are able to adapt in real-time to changing system conditions. ILP based solutions are able to achieve optimal task assignment, but cannot be used in dynamically changing conditions due to their computationally expensive nature. We evaluate the performance of our three dynamic algorithms using Qualnet and show that they have up to 88% longer system lifetime than the ILP based solutions.*

Keywords — Energy minimization, system life, task assignment, graph partitioning, mincut, distributed processing, sensors, healthcare, wireless communications.

1 INTRODUCTION

Recent years have witnessed the grow of two technologies: (1) wireless mobile systems able to provide wide coverage and seamless connectivity with personal devices, and (2) low-cost, miniature physiological sensors which can be organized in Body Area Networks (*BAN*) and provide continuous collection of personal data. The union of mobile systems and *BAN* enables the development of a large number applications in healthcare [1], gaming [2] [3], sports [4] and education [5].

BAN is a Wireless Sensor Network (WSN) whose nodes are worn by the user and whose job is to collect physiological signals [6]. Data from the body worn sensor nodes is forwarded to a sink node carried by the user which stores it locally [7][8] or sends it to a server [9][10]. Mobile sensing systems extend the *BAN* by adding a Local data Aggregator, *LA*, (i.e. a cell phone or a PDA) and a Back End server, *BE* [11][12][13]. Within a mobile sensing system, the *LA* acts as the interface between the user and the rest of the system, and provides enhanced computational and memory capabilities to analyze the data from sensor nodes, provide feedback to the user, and seamless connectivity between the *BAN* and the *BE*.

Typically data processing in mobile systems is centralized (all on the *BE*). In many applications it is not possible to always have reliable connectivity to the back end servers. Furthermore, since the cost of computing data locally is lower than wirelessly transmitting it [14][15], a more energy efficient approach is to distribute tasks among different devices (the Sensors of the *BAN*, the *LA* and the *BE*). However, efficient task allocation in a heterogeneous system is challenging due to performance and energy constraints of *LA* and *BAN*.

In this paper we present and compare three different dynamic task allocation algorithms: *DynAGreen*, *DynALife* and *DynAGreenLife* which address the following objectives respectively:

Minimizing the system energy – *DynAGreen*. System energy is defined as the total energy consumed by all mobile components (i.e. sensors and *LA*) in the system. Minimizing the system energy can result in a significantly different battery lifetime in heterogeneous devices due to the uneven workload distribution. Since the system lifetime is the minimum battery lifetime of all devices, minimizing system energy, while fair across nodes, may result in a node being depleted of its battery energy faster, resulting in a shorter overall system lifetime.

Maximizing the system lifetime – *DynALife.* System lifetime is defined as the time from the start of operation to the point in time when the first battery-operated device is out of energy. Due to heterogeneity of the system, each device can have a very different battery capacity. Thus, optimizing for system lifetime leads to significantly different task allocation as compared to the previous one.

Balancing the system energy and lifetime – *DynAGreenLife.* This objective balances the trade-off between minimizing system energy and maximizing system lifetime.

We demonstrate the effectiveness of our algorithms by comparing them with the state of the art Integer Linear Programming (*ILP*) based optimizations shown in Section 4. *ILP* solutions can find an optimal task assignment for smaller problem formulations; however, this comes at a prohibitive computational cost in dynamic scenarios and for larger problem sets. Similar approaches have been used by other publications, such as in [35]. We compare our dynamic approaches to the static *ILP* solutions and show in Section 6.1 that in static conditions our dynamic algorithms are within 0.001% of the optimal ILP solution. Furthermore, we provide a detailed comparison to a quasi-dynamic table based ILP solutions which provide an optimal solutions for when a mobile is near a base station, with *ILP-near*, far from the base station, with *ILP-far,* and where mobile frequently changes direction, with *ILP-flip-flop*. We show in Section 6 that our dynamic algorithms still outperform these table-based ILPs by as much as 88%.

The rest of the paper is organized as follows. Section 2 discusses the prior work in wireless healthcare. Section 3 provides a formal description of the system model and the task binding problem. Section 4 describes state of the art *ILP*-based static techniques for the energy management in wireless healthcare systems. Our dynamic assignment techniques are described in Section 5. Finally, we present the results in Section 6, and conclude in Section 7.

2 RELATED WORK

The combination of body area networks and mobile systems that allow ubiquitous connectivity have found recent application in wireless healthcare. Within the Mobihealth [16] European project, a complete end-to-end m-health platform for ambulant patient monitoring, has been deployed over UMTS and GPRS networks. The platform is based on a body area network which includes an ECG sensor, a respiration sensor, an activity/movement sensor and a pulse-oximeter. This network forwards the collected raw data to applications running on the backend. In [16], Dabiri et.al. have presented the Electronic Orthotic Shoes system as a means of ulcer prevention for patients suffering from neuropathy. There are many other systems such as Alarm-Net [18], SmartCane [19], I-Living [20], and PAMM [21] to aid the elderly in their day-to-day lifetime. Similarly, [19] is an example of a wireless preventive healthcare system.

Most state of the art solutions follow a sense-and-forward model where data is gathered by the sensors and sent to the back end server for processing. Previous work has shown that transmitting or receiving a bit wirelessly is significantly more expensive than processing the bit locally on a sensor node microcontroller [14][15]. Thus, the sense-and-forward model may lead to significantly lower system lifetime. Distributed task allocation is one way to extend the lifetime.

There is quite a bit of previous work on energy optimization in wireless sensor networks using dynamic task assignment. In [22], the static EcoMapS algorithm is proposed for energy constrained applications in single-hop clustered WSNs with homogeneous nodes, to map and schedule communication and computation simultaneously. EcoMapS schedules tasks with the minimum delay subject to energy consumption constraints. A quick recovery algorithm is executed to generate an alternative schedule if a sensor fails. This technique works for homogeneous sensor networks, but does not consider the case where nodes have different characteristics. An algorithm for task mapping and scheduling in multi-hop homogeneous WSNs with real-time guarantees is proposed in [23]. Sensor network nodes are grouped in homogeneous clusters which execute an application and communicate via a multi hop network. Thus, these results cannot be applied to sensor networks where nodes have heterogeneous capabilities and are organized in a hierarchical manner.

Although task mapping and scheduling techniques have been applied to wired and wireless sensor networks, very little has been done in wireless sensing systems with heterogeneous nodes organized hierarchically. In [24], a technique based on an offline Maximum-Flow Minimum-Cut followed by dynamic adjustment is proposed. This work schedules a set of tasks on a mesh network of nodes in order to minimize the total network energy consumption. Dynamic adjustments are applied to reduce the workload of nodes whose energy is below a fixed threshold. This solution makes the unrealistic assumption that there are no constraints on the energy consumption of the local aggregator. Furthermore, the time complexity of the proposed dynamic adjustment step can be more than exponential in the worst case scenario, which makes it infeasible for complex systems made up of a large number of low-power nodes.

Task assignment and scheduling for multiple devices are classical problems in traditional computation and parallel computing [25][26][27]. Stone [27] provided a graph-based partitioning solution to efficiently assign program modules in a two-processor, distributed computing system to minimize the total execution time of system tasks. In our earlier work we propose a static, ILP based solution that minimizes the maximum fraction of energy consumed with a specific task and device binding [36], and then follow up by designing a simple heuristic solution that maximizes the system lifetime in [37]. We also leveraged Stone's min-cut graph method [27] and applied it to task partitioning to optimize for the energy savings [38]. While both heuristic algorithms are compared to their representative ILP solutions, they are not compared to each other, and are not studied under more realistic conditions.

The work presented in this paper focuses on reducing the total energy consumption and maximizing lifetime of mobile sensing systems by performing dynamic task assignment and scheduling. Unlike most prior work in *WSN* and *BAN* task assignment, our strategy is able to work with a heterogeneous set of nodes with various performance and energy characteristics and varying communication capabilities. Our technique can adapt to changing system and workload characteristics at runtime, such as varying wireless channel conditions, processing load, network load, task arrival rates, and the available battery capacity of the devices. In this paper, we introduce a new objective in addition to the system energy and lifetime optimization: we balance the tradeoff between system energy and system lifetime using a graph based approach. We also propose an extension of our technique that can perform energy efficient task allocation for specialized task sets, which distribute their output to multiple dependent tasks (i.e. task out-degree greater than 1). This is particularly challenging since the energy costs associated with data transmission depend on the tasks allocation and are not known a priori. We provide a complete comparison between all three different objectives of our dynamic algorithms, and comparisons to ILP-based solutions for each objective along with a table-based ILP lookup method that provides a fast way to estimate a solution in dynamic conditions from pre-computed ILP results. Furthermore, we study in depth how our algorithms behave in more realistic dynamic scenarios by leveraging through Qualnet simulation [34] and show in Section 6 that our dynamic algorithms outperform the table-based ILPs by as much as 88%.

3 SYSTEM MODEL

The architecture of the mobile sensing system addressed in this work is shown in Figure 1. The system consists of three main components: Sensors, a Local Aggregator (*LA)*, and a BackEnd Server (*BE*). The sensors detect specific events on a human body and/or in the environment (e.g., low blood sugar, CO level, ozone level) and send processed and/or raw sample data to their respective *LA* wirelessly. No direct communication between sensors is available. Sensors can send and receive data only through the *LA*. The *LA* aggregates sensor data and may process the data before sending it to a centralized *BE*. It typically has multiple wireless technologies, such as Bluetooth and Zigbee to communicate with sensors, and WLAN and WWAN to communicate with the *BE*. The *BE* in turn receives the information from the *LA* and stores it in a database. The *BE* can also use the received information for generating real-time alerts. In this heterogeneous architecture, the devices can differ widely in their processing capabilities, battery charge, communication bandwidth, communication range, and the types of radios.

Tasks of a mobile sensing application can be modeled as a Directed Acyclic Graph (DAG) $G=(T, C)$ where node set T represents a set of n sensing/processing tasks, T_i: $i = 1, 2, \ldots n$, and C is a set of edges that represent communication tasks between nodes, C_{ij} : $i, j = 1, 2, \ldots n$. $C_{ij} \in C$ represents a precedence relation between tasks T_i , $T_j \in T$ and the data produced by task T_i should be communicated to T_j before T_j can start its processing. The weights w_{ij} on edge $C_{ij} \in C$ represent the amount of data that needs to be transmitted from task T_i to T_j. Tasks that do not have any predecessors are called source tasks, and tasks that do not have successors are called sink tasks. Generally, source tasks are used for sensing and the sink tasks are used for data logging at the *BE*.

Given a DAG and a set of heterogeneous resources, our goal is to map the given DAG onto the set of resources, R, such that i) the total system energy consumption is minimized; or ii) the system *lifetime is maximized; or iii) system energy and system lifetime are balanced*. In this paper we use the term resource to refer to nodes in a mobile sensing system: Sensors, LAs and/or BEs. The energy consumed by all the battery-operated devices (Sensors and *LA*) is the total energy consumption. The system lifetime is defined as the minimum of the battery lifetimes of all the mobile components carried by the same user as this is what is important from each user's perspective. Task allocation must consider the hierarchical and heterogeneous nature of the system, which requires that some tasks are bound to specific devices. For example, sensing can only be performed by the sensors of the *BAN* and not by the *BE*. We assume that all the tasks that a device can execute are precompiled and loaded in the device's memory. This reduces the cost and time associated with device reconfiguration.

Tasks of the *DAG* are further characterized by a set of variables and constants given in Table 1. A task *i*'s execution time is defined by t_{ir} and amount of data transmitted to task *j* is defined by w_{ij}. Table 1 also defines variables and constants (E_{Br} , P_{Rr} , P_{Tr} , P_{Er} , μ_{Rr} , μ_{Tr}) to characterize a resource *r*. These variables together define communication and computation energy cost of task *i* on resource *r*. For example, E_{Rir} defines the energy consumed by task *i* to receive its input data on resource *r*. This is calculated from resource *r*'s receive power P_{Rr}, its average receive data rate μ_{Rr} and time to receive task *i*'s input data from its predecessor task *j* is given by (w_{ij}/μ_{Rr}). The receiving energy cost of a task is nonzero only when the task's predecessor is bound to a different resource than that of the current task. Similarly E_{Tir} defines the transmit energy associated with task *i* on resource *r*. A task *i*'s computation energy is given by $P_{Er} \times t_{ir}$. The total energy E_r consumed by a resource *r*, is the sum of the computation energy of all tasks assigned on *r*, the energy to receive input of these assigned tasks and transmission energy to transmit the output of these tasks to its successor on other resources.

Additional communication tasks are added in the task graph for the case where a sensing or processing task is executed on a sensor and its successor task is mapped to the backend server. In this case, the data needs to be transferred through the local aggregator and the energy cost of this transfer needs to be accounted for. By binding the communication task to the local aggregator, we can calculate the energy cost of this transfer of data which would simply be the energy needed to receive and transmit the data on the local aggregator.

A sample DAG for activity detection application is shown in Figure 2. The system includes three accelerometers which are worn on hand, waist and leg to determine whether a person is sitting walking or running. The three sensing tasks act as source tasks. *Activity-Count1-2-3* are the successors of their respective sensing tasks. These processing tasks generate activity count from the accelerometer sensed data. *Acc-Correlation* is a processing task that combines the output of its three predecessors and determines the current activity of the person. *Activity-log* is a sink task bound to *BE* to store the activity log for future analysis by a medical professional, physical activity trainer, etc. In this task graph *ACC1*, *ACC2* and *ACC3* are the source tasks and so they do not have any predecessor. Activity Count1, Activity Count2 and Activity Count3 are the successors of ACC1, ACC2 and ACC3 respectively. Hence, $suc_{ACC1,ActivityCount1} = 1$, $suc_{ACC\ 2,ActivityCount2} = 1$, $suc_{ACC3,ActivityCount3} = 1$, $pred_{ActivityCount1,ACC1} = 1$, $pred_{ActivityCount2,ACC2} = 1$ and $pred_{ActivityCount3,ACC3} = 1$. Each of the tasks in a task graph has start time s_i, an end time τ_i and a deadline d_i associated with it. To maintain the task precedence, the start time of a successor should be greater than the end time of its predecessor. Thus, in this task graph, $s_{ActivityCount1} > \tau_{ACC1}$.

We next outline the *ILP* based optimization used to obtain the initial task assignment for our dynamic task allocation algorithms described in Section 5. The ILPs are also used for comparison to our dynamic solutions.

4 OPTIMAL STATIC TASK ASSIGNMENT

Integer Linear Programs (*ILP*) optimization is a state of the art technique that has been used to compute the optimal task assignment in smaller size wireless sensor networks [28]. Two main objectives are defined: i) *ILP*Green - minimizing system energy ii) *ILP*Life - maximizing system lifetime. These static solutions act as a baseline for comparison with the dynamic algorithms presented in this paper. Furthermore, they provide the initial task assignment to dynamic solutions.

*ILP*Green computes the most efficient task allocation that minimizes the total energy consumption of the system. It distributes the tasks among the different nodes in the system (backend, *LA*, and sensors) to make the entire system as energy efficient as possible. The format of the task graph is the same as the DAG previously described, except that we add an explicit communication task in between two dependent sensing/processing tasks to simplify the *ILP* formulation. Figure 3 shows a sample graph transformation. From the graph, we can determine each task's predecessors, successors, amount of data needed from previous tasks, and the amount of data it produces. *ILP*Green assigns tasks with the aim of reducing the total energy consumption, which may actually result in more work being assigned to a resource with relatively low battery charge. As a result, that resource would discharge earlier, ultimately resulting in shorter system lifetime.

We next define *ILP*Life that maximizes the system lifetime using a min-max formulation on the rate of energy consumption relative to the available battery capacity of each node. *ILP*Life has the same constraints, constants and variables as *ILP*Green but computes the optimal assignment with the goal of maximizing system lifetime. This approach balances the energy consumption of all resources so that each resource consumes approximately the same percentage of energy, since any resource consuming a higher percentage of energy than the others would deplete the battery capacity first, and shorten the overall system lifetime.The complete formulation of *ILP*Green and *ILP*Life with goals and constraints are given in Table 2 while all the variables and constants used are defined in Table 1. Among the resources *r*, resource *m* denotes the backend server and resource *m-1* denotes the local aggregator. We solve the above *ILP*s' for b_{ir}, the mapping of tasks i to resources r. Variable E_r represents the total energy consumed by a resource after tasks have been assigned to it. For *ILP*Green, our goal is to minimize the sum of energy consumed by all the resources except *BE* i.e. sum of all E_r. Our goal for *ILP*Life is to minimize the maximum percentage of energy consumption among resources. In Table 2, E_r/E_{Br} represents the percentage of energy consumed by a resource r.

The constraints of the *ILP* are described in Table 2. Constraint (a) ensures that tasks are assigned to only one resource. Constraint (b) states that a resource is not assigned to more tasks than the resource has energy for. Resource allocation constraints (c), (d), and (e) ensure that all predecessor tasks of a task are allocated to neighboring resources and that are able to communicate with each other. Constraints (f), (g), and (h) are related to timing and delays. Constraint (f) ensures that tasks do not start before their predecessor tasks have finished. Constraint (g) determines when a task finishes by calculating the time it takes to receive, execute, and send the data. The deadline constraint (h) makes sure that all tasks with deadlines finish on time.

In a practical system, we should be able to adapt to the various sources of runtime variations, such as wireless channel condition changes due to a person moving inside of a building, or task execution time changes due to more processing required in certain situation, or the introduction of new tasks. Changes in wireless channel conditions result in changes in communication energy cost, while changes in execution time affect computation energy cost. Since *LA* is a central component in our system, it can easily keep track of varying system parameters without too much communication overhead. Because of this, any dynamic task assignment algorithm should be run on *LA* at regular intervals, or whenever the system parameters change.

We use an open source *ILP* solver, *lp-solve* [29], to get the optimal task assignment for each of our task sets. Figure 4 shows the execution time of *lp-solve* on a 1.8GHz processor for various task sets and compares it to runtime of dynamic algorithms described in next section. As the number of tasks in a task set increases, the execution time of *lp-solve* increases exponentially and quickly becomes prohibitively high to execute on the *LA*. Because of this, the *ILP* based solution is not practical and cannot be deployed on the *LA*. To address this challenge, we have designed three fast task assignment algorithms. These algorithms have very low computational overhead compared to *ILP* as shown in Figure 4. While for more complex task sets the ILPs take close to 1000 seconds, the DynAGreenLife algorithm presented in this paper runs in under a millisecond, more than a million times faster. Hence the dynamic algorithms can be run frequently to find energy efficient task assignments in changing environments. The following section describes these algorithms in detail.

5 DYNAMIC TASK ASSIGNMENT

We have designed three fast and energy-efficient task assignment algorithms: *DynAGreen* to minimize total energy consumption; *DynALife*, to maximize the system lifetime; and *DynAGreenLife* which balances the two objectives. These dynamic algorithms run periodically on *LA* to find an energy-efficient task assignment. They monitor task energy consumption, node communication costs and battery charge, and then compute the task assignment that meets the objectives. We assume that the firmware on each sensor node and the application on the *LA* already include all the possible tasks that a device may execute. Thus, once a new task assignment is computed, a simple reconfiguration message is sent to the sensors to activate an already-present task. This approach reduces the time and energy cost associated with device reconfiguration and has been used in previous work for wireless sensor networks [29].

Each algorithm uses a hierarchical flow graph partitioning technique to partition tasks between different components (*BE*, *LA*, sensors) which follows a common pattern composed of two steps. In the first step, a flow graph constructed from a given task graph is partitioned between a server with unlimited energy supply, *BE*, and a super- node referred as *BAN* that has both the *LA* and the sensors. Weights on the edges in the flow graph represent communication and computation energy costs of performing a task on *BAN* or *BE*. Partitioning is done using a maxflow/mincut algorithm. The minimum weight cutset represents the energy cost if tasks are partitioned by the cutset. This provides the partitioning of tasks that minimizes the energy consumption between the *BE* and the *BAN*. In the second step, we partition the tasks within the *BAN* between the *LA* and the sensors to minimize the utilization of energy within the *BAN*. We use the implementation of mincut/maxflow algorithm proposed in [31] to implement our two stages of the hierarchical graph partitioning technique.

The following subsections first describe the energy minimization algorithm *DynAGreen* followed by *DynALife* for system lifetime maximization and then we introduce *DynAGreenLife* algorithm which balances both objectives.

5.1 DynAGreen Algorithm

The *DynAGreen* algorithm minimizes the total system energy consumption. Figure 5 shows the outline of the *DynAGreen* algorithm. The step numbers listed below correspond to those in Figure 5.

I. **Initialization:** Energy cost parameters (E_{CPUa}, E_{CPUi}) and communication energy cost parameters (E_{tx} , E_{idle}, E_{rx}) for all resources are initialized/updated. Here $E_{CPUa}(r)$ and $E_{CPUi}(r)$ represent the energy consumed by a CPU in active and idle states respectively, E_{tx}, E_{rx}, and E_{idle} represent the energy consumed by radio in transmit, receive and idle states respectively. In the first run of the algorithm they are initialized with default values, while in the subsequent runs they are updated based on the radio interface monitoring.

II. ***BAN-BE Partitioning*:** Tasks are partitioned between *BAN* and *BE* so that the total of computation energy cost of performing tasks in *BAN* partition and communication energy cost of sending output of those tasks to *BE* is minimum. To achieve this objective we transform the task graph into a flow graph with communication and computation energy costs as flow values, *BAN* as a source node and *BE* as sink node so that the minimum weight cutset represents the total of computation and communication energy cost for the partition. Following is a formal description of this transformation:

- A flow graph $G' = (T', E')$ from a given task graph $G = (T, E)$ is created by adding *BAN* and *LA* nodes to *T* and adding E_{BAN} and E_{BE} edge sets to *E*. E_{BAN} and E_{BE} represent the energy costs of computation while the edges in E represent the energy costs of communication. Formally, $T' = T \cup \{BAN, BE\}$ and $E' = E \cup E_{BAN} \cup E_{BE}$. The *BAN* node represents the sensors and *LA*.

- *Defition of E_{BAN}:* E_{BAN} is a set of edges from *BAN* to the each node *t* in *T*. The weight of edge (*BAN*, *t*) corresponds to the computational cost of task *t* on *BE* and it is defined by $w(BAN, t)$ in Table 3. The rationale behind this weight assignment is that if a mincut includes edge (*BAN*, *t*) then it means that *t* is in *BE* partition and the computation cost of performing this task on *BE* is added to the weight of the cutset. Being powered from power line, *BE* has unlimited energy, so in Table 3 the computation energy cost of performing a task on *BE* to is set to 0. If a task *t* is bound to a *BAN*, the minimum weight cutset should not include edge (*BAN*, *t*). We achieve this objective by setting the weight of the edge (*BAN*, *t*) to ∞ (see Table 3).

- *Defition of E_{BE}:* E_{BE} is a set of edges from each node *t* in *T* to *BE*. The weight of edge (*t*, *BE*) is equal to the computational energy cost of executing task *t* on *BAN* as defined by $w(t, BE)$ in Table 3 ($E_{CPU}(t, LA)$). The rationale is the same as for (*BAN*, *t*) weight assignment. If a mincut includes edge (*t*, *BE*) it means that *t* is in the *BAN* partition and the computation energy cost of performing this task on *BAN* is added to weight of the cutset. If the task is bound to a *BAN* we set its weight to 0. If the task is bound to *BE* we set its weight to ∞ .

- *Edge*(t_p, t_s) in *E* represent the inter-task communication link between the predecessor task t_p and its successor task t_s. The weight of edge (t_p, t_s) is equal to communication cost of sending t_p's output from *LA* to *BE*. This value is defined in Table 3 as $w_{wan}(t_p, t_s)$, where $E_{radio}(t_p, LA)$ is the communication energy consumed by task t_p on resource *LA*.

III. ***Min-cut partitioning*****:** We find the minimum weight cutset to partition the constructed flow graph into *BAN* and *BE* partitions. In [27] Harold Stone proves that such a minimum weight cutset provides the optimal task assignment for a two processor system. Thus, our *BAN-BE* task partition results in the minimum energy cost.

IV. ***Task allocation*****:** Since we found the minimum energy cost partition between *BAN* and *BE*, tasks in *BE* partition are assigned to *BE* while tasks in *BAN* partition need to be assigned between Sensors and *LA* in the subsequent steps of the algorithm. Define $T_{BAN} \subseteq T$ as a set of tasks in *BAN* partition and $E_{BAN} \subseteq E$ as a set of edges among tasks in T_{BAN}.

V. ***Sensor-LA Partition:*** All the sensors are collectively represented by a single source node *Sensors* and *LA* is represented by a single sink node *LA* in *Sensors-LA* flow graph. All tasks trace back to one or more sensing tasks by following their predecessor chain. These sensing tasks are pre-assigned to sensors or *LA*, depending on what device they are bound to. For example, a processing task that receives its input from more than one sensor cannot be assigned to a sensor since in our system sensors do not communicate with each other directly. Such a task is assigned to *LA* or *BE*. By assigning appropriate weights on the edges in flow graph, we ensure that such tasks cannot be in the *Sensors* partition. This also implies that if a task is in the *Sensors* partition, it should receive its input from only one sensor. Even though *Sensors* node in the flow graph represents multiple sensors, after finding mincut, we can assign tasks in *Sensors* partition to the appropriate sensors. Similar to *BAN-BE* partitioning, we partition tasks between Sensors and *LA* so that the total of computation and communication energy cost of the system is minimized. The minimum weight cutset of *Sensors-LA* flow graph represents the total of computation and communication energy cost for the partition. Following is a formal description of this transformation:

- *Create Sensors-LA partition*: We create a new flow graph G″ for *Sensors-LA* partition using the tasks in BAN partition along with *Sensors*, *LA* and a set of proxy nodes. Proxy tasks, T_{proxy}, are bound to *LA* and represents the communication energy cost of sending data from *LA* or sensors (through the *LA*) to *BE*. A proxy task for each task in T_{BAN} whose successors are assigned on *BE* is added. The edges of this new graph consist of the following edge sets: $E_{sensors}$ and E_{LA}, which represent the computation energy cost of the tasks on sensors and *LA* respectively, and E_{proxy}, which gives the communication energy cost of tasks. Formally G″ = (T″, E″), where T″ = ($T_{BAN} \cup T_{proxy} \cup \{Sensors, LA\}$) and E″ = ($E_{BAN} \cup E_{Sensors} \cup E_{LA} \cup E_{proxy}$).

- *Definition of $E_{Sensors}$*: $E_{Sensors}$ is a set of edges from sensor nodes to each node *t* in $T_{BAN} \cup T_{proxy}$. Its weight is equal to the computational energy cost of running task *t* on *LA* (*w*(*Sensors*, *t*) in Table 4). The rationale is that if a mincut includes edge (*Sensors*, *t*) then it means that *t* is allocated on *LA* and the computation cost of performing this task on *LA* is added to weight of the cutset. If a task *t* is bound to a sensor, we ensure that a minimum weight cutset does not include the edge (*Sensors*, *t*) by setting

$w(Sensors, t) = \infty$ (see Table 4). Similarly if a task t is bound to LA, we set $w(Sensors, t) = 0$.

- *Definition of* E_{LA}: E_{LA} is a set of edges from each node t in $T_{BAN} \cup T_{proxy}$ to LA whose weight is equal to the computational energy cost of running the task t on the *Sensors* ($w(t, LA)$ in Table 4), where $E_{CPU}(t, Sensor)$ is the computation energy cost of running the task t on a sensor). If a task receives input from multiple sensors, it is bounded to LA and has $w(t, LA) = \infty$ to ensure that this edge does not become part of minimum weight cutset. If a task t is bound to a sensor, $w(t, LA) = 0$ to ensure that the edge (t, LA) is part of the minimum weight cutset. For tasks t that are not bound to a sensor or LA, we find the sensor generating its input (by traversing task's predecessor chain) and we use that sensors processing costs to set the computation energy cost of the edge.

- *Definition of* E_{proxy}: E_{proxy} is a set of edges (t_p, t_s) where $t_p \in T_{BAN}$ and $t_s \in T_{proxy}$. E_{BAN} is a set of edges (t_p, t_s) where $t_p, t_s \in T_{BAN}$. The weight of an edge edges $(t_p, t_s) \in E_{BAN} \cup E_{proxy}$ represents the communication cost of transmitting t_p's output from its source. In Table 4 is given by $w_{ban}(t_p, t_s)$ where $E_{radio}(t_p, Sensor)$ is the communication energy consumed by task t_p on the resource Sensor.

VI. ***Min-cut partition for Sensor & LA*:** We find the minimum weight cutset to partition the constructed flow graph into *Sensors* and *LA* partition. This task partition results in a task assignment that has minimum energy cost. Sensor-LA partitioning is explained in detail below.

VII. ***LA & sensor task allocation*:** Tasks in *Sensor* partition are assigned to respective sensors and tasks in *LA* partition are assigned to *LA*. Even though Sensors the node in flow graph represents multiple sensors, after finding a mincut, we can assign the tasks in Sensors partition to the appropriate sensor by traversing the task predecessors chain until we find the source sensing task. The task is assigned to the same sensor node.

VIII. **Parameter updates:** To detect the changes in system characteristics, communication parameters such as transmit power and time spent in various states (receive, transmit, idle, sleep) by the radio on each resource are measured and updated. This measurement is done over Least Common Multiple (LCM) of the period of all tasks in the task graph. The algorithm restarts whenever significant change (in our case 30%) in the communication cost parameters is detected.

We use the task set initially introduced in Figure 2 to provide an intuitive explanation of the algorithm, starting with *BAN-BE* partitioning and then *Sensors-LA* partitioning steps.

***BAN-BE Partition**:* Figure 6 shows the flowgraph G' for the *BAN-BE* task partition constructed from the graph shown in Figure 2. Notice that we added two new nodes *BAN* and *BE* to the original graph. All the edges from *BAN* to the tasks have the computation cost of running tasks on *BE* as their weights (see Table 3). We set the computational cost of running a task on *BE* to zero since it does not affect battery lifetime of sensors or *LA*. For example, the edges from the *BAN* node to tasks *3, 4, 5, 6 and 7* have weight of 0 in Figure 6. If a task is bound to a specific sensor or *LA*, we make sure that the edge between *BAN* and the task does not become a part of minimum cutset. This is ensured by assigning a weight of ∞ to these edges. Since in our example tasks 0, 1, and 2 are bound to their respective sensors, weights of edges from *BAN* to *tasks 0*, *1*, and *2* are set to ∞.

All the edges from tasks to *BE* have the computation costs of running them on *LA*. Their weights are set as shown in Table 3. For example, edge from task 3 to *BE* has energy cost $E_{CPU}(3, LA)$. For the tasks that are bound to either sensors or *LA*, we ensure that the edge from that task to *BE* is a part of minimum cutset and hence we set it 0 in Table 3. If a task is bound to *BE* we ensure that an edge from given task to *BE* is not part of minimum cutset and hence we set the weight to ∞ as in case of *Activity Log* task in this example.

The weights of the edges between the tasks represent the communication cost. For example, the weight of the edge between *task 0* and *task 3* represents the amount of energy required to send output of *task 0* from *LA* to *BE*. Table 3 summarizes how these weights are defined. We use *LA*'s wide area network radio energy parameters for these cost calculations, since we are trying to find the tradeoff between computing a given task on *BAN* and then sending its output to *BE*, or sending the input of given task from *LA* to *BE*. In our specific example, communication cost of sending input of *task 3* to *BE* (3.145) is quite high compared to summation of communication cost of sending its output to *BE* (1.51) and computation cost of doing task on *LA* (0.231). Figure 6 shows the minimum weight cut set for the example task set. This provides the energy cost optimal assignment for *BAN-BE* partition as per [27]. As a result of this, task 7 is assigned to *BE*.

Sensor-LA Partition: Figure 7 shows the flowgraph G'' for the *Sensors-LA* task partition constructed from graph *G'*. Nodes representing the *Sensors* and the *LA* are added to the tasks graph. In our example, we also add an *ACC Correlation Proxy task 6* because its successor is assigned on *BE*. Therefore, if *task 6* is assigned to a sensor we take into account also the fact that the LA should forward the task output to the BE. Given the hierarchical nature of our system which do not allows inter-sensor communication link, we cannot assign a task to a sensor if it receives its input from multiple sensors sources. In our example *ACC Correlation* is bound to *LA*. The edges weights are defined in Table 4. Figure 7 shows minimum weight cut set for Sensor-*LA* partition. All tasks in *LA* partitions are assigned to *LA*. Note that if a task and task's proxy are both in *LA* partition, we ignore the proxy task since it is only required if its corresponding task is not assigned to *LA*.

In Figure 4, we compare the execution time of *ILP*-Green and *DynAGreen* when run on a PC. The computational time of *ILP* is three order of magnitude higher (seconds) than *DynAGreen*'s execution time (milliseconds). *ILP*-Green execution time increases exponentially with a number of tasks in the task graph as indicated by Figure 4. Because of this low computational overhead compared to *ILP*-Green, our approach enables frequent adaptation to dynamic changing system parameters.

5.2 DynALife Algorithm

In this section we describe a fast and energy-efficient task assignment algorithm called *DynALife*, designed to meet the objective of maximizing the system lifetime. System lifetime is defined as a minimum of battery lifetime of all devices in the mobile sensing system. Cellphones and sensors can have different battery charge capacity and different rate of energy consumption. To maximize the system lifetime, we need to assign tasks such that node with the least battery charge gets the least amount of work. *DynAGreen* algorithm does not consider the remaining battery charge of a device while assigning tasks and hence minimizing energy consumption using *DynAGreen* algorithm does not always maximize the system lifetime. For example, let's say at the time of assignment shown in Figure 9 computed by *DynAGreen*, *LA* has only 10% battery charge remaining while all sensors have 80% of their battery capacity remaining. In this scenario we can reduce the workload on *LA* by assigning *Activity Count* tasks to sensors instead of *LA* and thus increase the system lifetime.

Figure 10 outlines *DynALife* heuristic algorithm. We assign the tasks one at a time starting from the source tasks. In each case we select a task assignment that maximizes the system battery lifetime compared to all other task assignments possible at a given stage that also meet the task deadline. We perform this operation iteratively until all tasks are assigned. After the initial assignment we iterate over the assignments and reexamine each task's assignment to see if an alternative assignment can improve the system lifetime. We end the algorithm when further iteration does not improve the system battery lifetime. Following are the definitions of the variables used by the algorithm: T : set of n tasks in the system, $P \subseteq T$: set of pre-allocated tasks, $A \subseteq T$: set of already assigned tasks, $R \subseteq T$: set of unassigned tasks, and $E \subseteq T$: set of eligible tasks. Algorithm steps are shown in Figure 10. For more details see [37]. The complexity of the entire *DynALife* algorithm is $O(n^3 log(n))$.

DynALife targets the most critical energy resource node (either the local aggregator or one of the sensors) in a system with M nodes, and ensures that this critical node is assigned the least amount of tasks in the system so that its lifetime is elongated as much as possible using the min-max formulation. However, this process neglects how energy is distributed between the remaining (M-1) nodes. Thus, we need a dynamic task assignment algorithm that maximizes the system lifetime while consuming less energy. To address this challenge we designed a third algorithm, which we call *DynAGreenLife*, that balances both system lifetime and system energy consumption.

5.3 *DynAGreenLife Algorithm*

In this section we describe *DynAGreenLife* that jointly optimizes for both system energy consumption and system lifetime. *DynaGreenLife* effectively attempts to minimize a weighted energy objective function $\sum(w_r * E_r)$ where w_r is proportional to the *reciprocal* of $1/E_{Br}$. When $w_r = 1$, then the objective function for *DynaGreenLife* reduces to that of *DynaGreen*. When w_r is 0 for all nodes except the one which has the fastest battery drainage (for which $w_r = 1/E_{Br}$), then the objective function reduces to the min-max formulation for the objective function of *DynaLife*. *DynAGreenLife* has exactly the same steps of *DynAGreen* except for step 5, the *Sensors-LA* partition. In the first step we initialize or update the energy cost parameters. In the second and third step, we create *BAN-BE* flow graph and find the mincut. In the forth step, we assign the tasks in *BE* partition to *BE*. In step 5, which differs from *DynAGreen*, we create *Sensor-LA* flow graph with different edge weights compared to *DynAGreen*. We obtain the weights of an edge by multiplying its *DynAGreen* weight with a factor which depends on the relative battery charge of the sensor in question and the *LA*. Equations (1)(2)(3) define the new weights of edges in the flow graph utilizing the definitions of weights in Table 4.

$$w'(Sensor, t) = w(Sensors, t) * \frac{E_{bat}(s) + E_{bat}(LA)}{E_{bat}(LA)} \tag{1}$$

$$w'(t, LA) = w(t,\ LA) * \frac{E_{bat}(s) + E_{bat}(LA)}{E_{bat}(s)} \tag{2}$$

$$w'_{ban}(t_p, t_s) = w_{ban}(t_p, t_s) * \frac{E_{bat}(s) + E_{bat}(LA)}{E_{bat}(LA)} \tag{3}$$

Equation (1) defines the weight of the edge from sensor node *s* to a task node *t*. Here *s* is the sensor node which runs the source sensor task for task *t*. Equation (2) defines weight of the edge from a task *t* to *LA* in the flow graph. To understand the effect of battery charge, consider a scenario in which sensors' battery has lower charge than *LA*'s battery. In this case *w*(*Sensors*, *t*) is multiplied by the smaller factor compared to *w*(*t*, *LA*), resulting in relatively lower weight to edge from Sensors *s* to task *t*. This results in increasing chances of *t* belonging to *LA* partition. Equation (3) defines the weight of edge (t_p, t_s) from a predecessor task t_p to successor task t_s. Since we increase the weight of communicating edges as well, this only favors the assignment of a task to *LA* if the cost of sending tasks from sensor to *LA* does not result in higher sensor energy drain.

We use the same task set example shown in Figure 2 as we did in *DynAGreen* algorithm to explain this algorithm in detail. The algorithm first performs *BAN-BE* partition and then *Sensors-LA* partition. The *BAN-BE* partition is the same as *DynAGreen* and is shown in Figure 6. However, for step 5, the *DynAGreenLife* computes the weights as shown in equations (1), (2), and (3). In our example, *LA* has critically low battery so the multiplication factors of the weight between sensors and tasks are higher than the one between tasks and *LA*, $\frac{E_{bat}(s)+E_{bat}(LA)}{E_{bat}(LA)} > \frac{E_{bat}(s)+E_{bat}(LA)}{E_{bat}(s)}$. As a consequence, a mincut does not pass through the edges from the sensors and more tasks are assigned on sensors. In Figure 9, the weights of edges from *task 0* to *task 3*, *task 1* to *task 4* and *task 2* to *task 5* are higher than the weight on edges from tasks 3, 4, 5 to *LA* node. This helps to assign four tasks to our most critical resource *LA*, by avoiding a cut on edges from task 0 to *task 3*, *task 1* to *task 4* and *task 2* to *task 5*. In Sensors partition we find a sensor node for a task by traversing its predecessor task chain until we find the source sensing task. We assign the given task to same sensor node as its source sensing task. For example, for task 3, task 0 is its source sensing task and hence task 3 is assigned to the same sensor as *task 0* if *task 3* is in sensor partition.

5.4 Handling Multiple Outbound Edges

The proposed algorithms are extended to consider the case where a task has an outbound degree greater than one. For example, consider the task graph presented in Figure 8. If the tasks T_B and T_C have the same input from task T_A, than the data sent by the sensor node to the *LA* is shared between both task T_B and T_C so the communication cost should be changed accordingly. This may change the optimal task allocation, especially in the case of complex tasks that operate on small amount of data but require a large number of instructions. In this case the reduced cost of communication may result in a different task allocation (i.e. on the *BE*, instead than on the *LA*). To solve this problem we use an iterative approach. Figure 8 shows how our technique behaves on a sample dataset where the output of one task has an out degree equal to 2. In this example we use *DynaGreen* algorithm, but our technique applies as well to the other algorithms we presented.

1. Perform one round of dynamic task assignment using one of the techniques described in the previous sections (Figure 8.a).

2. Check the task allocation results to detect if multiple tasks which share the same input data are allocated on the same device (i.e. the *LA* or *BE*) and receive their input from a task executed on a different device. For example, in Figure 8.a, tasks T_B and T_C are executed on the *LA* and receive their input from a task T_A which is executed on a sensor node.

3. Reduce the cost of communication between the tasks to consider the single transmission of the data, and evaluate the energy cost of this partition. If the data is shared among *N* devices, the communication cost is reduced to 1/*N* of the total cost. For example, if we consider the task graph presented in Figure 8, the communication cost between T_A, T_B and T_C is halved (Figure 8.b).

4. Perform a second round of dynamic task assignment with the new communication cost (Figure 8.c).

5. If the second round of task assignment results in a different assignment than the one obtained during the first round, update the communication costs between the tasks and evaluate the energy cost of this partition. For example, in Figure 8.b tasks T_B, T_C and T_D are now allocated on the *BE*. In this case an extra task (Proxy) is generated and allocated on the *LA* to reflect the fact the sensors cannot communicate directly to the *BE*. The communication cost between the *LA* and the *BE* are also halved to reflect that the data is shared between the two tasks, T_B and T_C.

6. Compare the energy cost of the partitions obtained at steps 3 and 5 and choose the one with the lower energy cost.

Table 6 shows the energy costs for the sensor node and the *LA* for the two solutions in Figure 8b and c. The latter task assignment is selected since it results in lower power consumption for the *LA*.

6 RESULTS

In this section we evaluate our dynamic task assignment techniques in terms of their effect on both the system's battery lifetime and energy consumption. We demonstrate results for different wireless health care system task sets, and analyze the performance of the *DynAGreen*, *DynALife* and *DynAGreenLife* algorithms in comparison with each other and with the static solutions given by *ILPGreen* and *ILPLife*. We also demonstrate the ability of our algorithms to adapt to changing system conditions. We show how *DynAGreenLife* algorithm balances both system lifetime and system energy by comparing it with *DynAGreen* and *DynALife* algorithms.

6.1.1 Experimental Setup

We use three task graphs for our experiments, described in more detail in Table 7. These task sets are from real applications used in the preventive healthcare system PALMS [32], whose objective is to advance the exposure biology research by developing new methods of physical activity data capture and analysis over longer periods of time [33]. Each task has an arrival rate, a deadline that is equal to the task period, number of instructions to be executed, and the total size of the output. We use Qualnet [34], a state of the art discrete event wireless network simulator, for our experiments. Qualnet provides accurate wireless channel models, a variety of wireless protocols along with the communication energy consumption, battery and mobility models. We add a model for computational energy consumption and use CPU current loads appropriate for MicaZ and Intel XScale processors. Sensor nodes are modeled as MicaZ nodes with Zigbee (802.15.4) radio. The *LA* is modeled as a UMTS-UE (User Equipment i.e. Handset) with a Zigbee radio interface and CPU running at 400MHz. We use Qualnet's built in energy model for the MicaZ node and specify a constant transmit power of 3dBm. For UMTS, we leverage a default energy model and specify -10dBm as the minimum transmit power and 30dBm as maximum transmit power. UMTS protocol uses dynamic power control algorithm and sets the radio transmit power depending on channel condition. We used 2.4GHz carrier frequency for UMTS and 905MHz carrier frequency for Zigbee.

The handset is connected to the *BE* via the UMTS network and to the sensors via Zigbee radio. Tasks communicate via UDP if any of the successor tasks are not assigned on the same device. We implement the logic for periodic task execution, data transmissions, radio link parameter measurements and reporting, task assignment control messaging and execution of the dynamic algorithms at the application layer.

6.1.2 Performance in Static Conditions

In the initial set of experiments, we assume that the system parameters such as the arrival rate of tasks, computational complexity, and wireless channel conditions remain constant. We use 100 mAh for sensors and 300 mAh for the cellphone battery energy so that the experiments are of appropriate length. We compute the system lifetime achieved by task assignment given by both the *ILP*s and our three dynamic algorithms and compare it to streaming all the data to BE for processing (All-On-BE).

In static conditions, both *ILP*s come up with the same task assignment for our task graphs. In Figure 11 we show the percentage improvement in system battery lifetime based on the task assignment, as determined by both *ILP*s i.e. *ILPGreen* and *ILPLife* with respect to *All-On-BE*. For the *All-Vital* task graph, that has a higher number of processing tasks, both the *ILP*s perform 60% better than *All-On-BE*. On average, the *ILP* improves the system battery lifetime by 37%. We ran our dynamic algorithms using exactly the same setup described for the ILPs, and found results to be within 0.001% of the ILP results shown in Figure 11, so this figure can be used to represent both dynamic and ILP results in static conditions for our task graphs. Our dynamic algorithms perform nearly optimal in these static scenarios.

In the next set of experiments we demonstrate the ability of our dynamic algorithms to adapt to the different amount of battery charge and varying consumption rates. In contrast to previous experiments, now we provide higher battery charge for the sensor (300 mAh), but a smaller amount for the *LA* (100 mAh). Figure 12 shows the battery lifetime improvement obtained by the *ILP*s and the *DynA* family of algorithms compared to the *All-On-BE* strategy. It is evident from these experiments that task assignment techniques make a large difference (of up to 140%) in battery lifetime compared to following a simplistic approach of performing all processing on the *BE*. The results also show that *ILPLife*, *DynALife*, and *DynAGreenLife* achieve a longer system life than *ILPGreen* and *DynAGreen*. This is because the task partitioning of *ILPGreen* and *DynAGreen* is purely focused on minimizing the system energy consumption without taking any consideration of the available battery charge of the system devices. *ILPLife* has a lower system lifetime compared to *DynALife* and *DynAGreenLife*. This is due to the fact that *DynALife* and *DynAGreenLife* periodically perform task repartitioning so that the task allocation changes to better optimize for the system lifetime. Depending on the tasks assigned, some nodes may deplete their energy more than desired. Hence a periodic repartitioning is necessary for a better solution.

We compare the system battery lifetime and system energy consumption obtained by *DynAGreen* and *DynALife* to that given by *DynAGreenLife* in Figure 13 and Figure 14. As expected *DynAGreen* is not able to extend the battery lifetime of the system as much as *DynALife* as it does not take into consideration the battery levels of the mobile devices. Both *DynAGreenLife* and *DynALife* take into consideration the remaining battery charge and thus the rate of energy consumption in addition to computation and communication cost. *DynAGreenLife* performs up to 43% better than *DynAGreen* in terms of system battery lifetime. *DynAGreenLife* performes slightly better than *DynALife* in some cases. *DynAGreen* uses a heuristic approach, assigning one task at a time instead of considering all the tasks or a chain of tasks for assignment. On the other hand, *DynAGreenLife* is a graph based solution that uses mincut/maxflow algorithm to partition tasks. Interestingly enough, although *DynAGreenLife* attempts to simultaneously satisfy objectives associated with minimizing the system energy and maximizing the system lifetime, it does as well or better with regard to maximizing system lifetime compared to *DynALife*. On the other hand, as *DynAGreen* is a greener solution whose goal is to minimize the energy consumption, it performs better than *DynALife* or *DynAGreenLife* in terms of energy savings as shown in Figure 14. When we compare *DynAGreenLife* and *DynALife*, both perform similarly in terms of battery lifetime but *DynAGreenLife* consumes up to 40% less energy compared to *DynALife* algorithm. To further understand these results, consider the different assignment given by both *ILP*s and dynamic algorithms for Activity Detect task set in the scenario where *LA* has a critically low battery level (100mAh) compared to the energy available for sensors (300mAh). For this task set and given the same set of static conditions, *ILPGreen* and *DynAGreen* give the same task assignment. Both the algorithms assign all processing to *LA* as this assignment minimizes the system energy consumption. On the other hand *ILPLife* and *DynALife* assign all three *Activity Count* tasks to respective sensors. To extend the system lifetime, this assignment reduces the processing load on *LA* while increasing the load on sensors even though assigning tasks on sensors results in the higher system energy consumption. As we have seen in results this achieves the objective of increasing the system lifetime at the cost of increased system energy consumption.

6.1.3 Dynamic Adaptability

In the following set of experiments we evaluate changes to various runtime characteristics and demonstrate the capability of our dynamic algorithms to change the task assignments dynamically to optimize the system lifetime and system energy consumption. We assume that the battery level is at 300 mAh for each sensor and that the cellphone has critically low battery level of 100 mAh. We show the improvement in system energy consumption using the dynamic task assignments given by *DynAGreen* with respect to the static assignment obtained by *ILPGreen* during initial setup. Similarly we show the improvement in system lifetime obtained by *DynALife* compared to *ILPLife*.

We measure various radio link parameters such as transmit power and time spent in various states (receive, transmit, idle, sleep) by each radio to detect the changes in system's characteristics. This measurement is done over a period in which tasks execution repeats. We use Lowest Common Multiple (LCM) of the period of all tasks. This window gives us a reproducible workload on each resource so that comparison of the various costs is fair. For our task sets LCM is around 1 minute. We also use filtering to smooth out unusual spikes in measurements. We run our dynamic algorithms to compute the task assignment only if the change in system parameters is more than 30% compared to the last run of algorithm. This threshold can be changed for different system implementations. We selected this threshold empirically for our simulation environment.

We divide our experimental scenarios in following two categories. 1) Illustrative scenario with defined mobility, defined workload changes, and only one base station on a smaller terrain area. 2) Urban scenario with random mobility, multiple base stations and large terrain area. In all these scenarios, we show that our dynamic algorithm adapts to changing system parameters quickly and results in better performance compared to optimal but static solution obtained using *ILP*s.

6.1.4 Illustrative Scenario

We simulated a typical day of the user to illustrate how our dynamic algorithms adapt to changes in wireless channel conditions and *LA* workload. We assume that the user's home is away from the base station and thus wireless link conditions are relatively poor with lower effective data rates compared to the case when the user is at work. The initial assignment is computed with the link conditions observed at home. If the user is mobile during the day and gets closer to the base station (e.g. at work, hospital, school), the wireless link conditions improve. Our proposed dynamic algorithms take advantage of these improving link conditions and allocate some of the processing tasks on *BE* instead than on *LA*. We assume that the user is at home at the start of the experiment and reaches very close to the base station after three hours. The user stays near the base station for 4 hours and comes back home in similar fashion. Since we maintain various link-related parameters for each mobile resource in the system, the *LA* detects these changes and updates the task assignments. The cost of UMTS transmission can increase by a factor of 10 depending on the user's distance from the base station.

In Table 8 we compare our *DynAGreenLife* algorithm with static assignments given by *ILPGreen* and *ILPLife*. When a user moves closer to the base station, our dynamic algorithms detect the reduced communication cost for *LA* to *BE* transmission and activates the processing tasks on *BE* instead of on *LA* if that results in lower system energy consumption or higher system lifetime. *DynAGreen* improves system lifetime for all task sets compared to *ILPGreen* by 14% on an average .It even improves system lifetime compared to *ILPLife* except for *Activity Detection* task by taking an advantage of new lower communication cost. Similarly, *DynALife* improves system lifetime up to 50% compared to *ILPGreen* and *ILPLife*. *DynAGreenLife* has a similar improvement as *DynALife* except for the case of *Activity Detection* task set. In this case it improves the system lifetime compared to *ILPGreen* by 25% while reduces system lifetime compared to *ILPLife* but gains in terms of the system energy consumption as shown in Figure 16.

Figure 15 and Figure 16 show the effectiveness of *DynAGreenLife* in balancing system lifetime maximization and system energy consumption minimization. *DynAGreenLife* obtains up to 13% better system lifetime compare to *DynAGreen* and up to 78% better system energy savings compared to *DynALife*. For *Activity Detect* task set, *DynAGreen* assigns all processing tasks on *LA* when user is away from the base station and it assigns all processing tasks to *BE* when user is closer to the base station. These assignments result in lowest system energy consumption. *DynALife* on the other hand considers the very low battery charge of *LA* compared to sensors and assigns all single source processing tasks to respective sensors to reduce the energy consumption of *LA*. For multi-source processing tasks *DynALife* keeps them on *LA* when *LA* is away from the base station and activates them on *BE* when *LA* gets closer to base station. *DynAGreenLife* strikes a balance between the two objectives. It generates the same assignment as *DynALife* when the user is away from the base station and it generates the same assignment as *DynAGreen* when the user is closer to the base station resulting in a nice balance of the system lifetime and system energy consumption.

Dynamic Load Balancing: Dynamic load balancing is required in the system as the processing complexity of a task may change depending on the data itself, or due to increased load on a resource in the system such as the *LA* (a phone serving as an *LA* may have to process other tasks). For example, if the sensed information is static or pseudo-static, the complexity of the tasks can be low based on the prior processed information. In our experiments we increase the execution time of processing tasks by a factor of 2 after 6 hours and return back to normal for the next 6 hours to simulate the two levels of processing load. We assume the same setup as in the previous experiment except for the fact that *LA* is not mobile. The *DynA* algorithms detect this variation in load and quickly re-compute the new task assignment in response. In Table 9 we compare the percentage improvement in system lifetime obtained by all three dynamic algorithms relative to *ILP*Green and *ILP*Life for change in execution time of the processing tasks. Our dynamic algorithms detect the increased computation cost and activate the processing tasks on the *BE* instead of *LA* if the computation cost is higher than the communication cost. *DynAGreen* improves system lifetime for HR+Activity task sets compared to *ILP*Green by 10%. Similarly, *DynALife* improves system lifetime up to 51% compared to *ILP*Green and *ILP*Life.

6.1.5 Urban Scenario

In this section we include a larger terrain with multiple base stations and random mobility of the users to demonstrate that our algorithm can handle a more complex, real-life urban situation. For our experimental study, we created a 4900 meter x 4900 meter terrain with four base stations in Qualnet simulator as shown in Figure 17. During the simulations the user moves randomly in this terrain in any direction with one of three speed ranges to simulate the user moving in car, bike and walking. Moreover, the user may pause for a predefined duration (15 min or 45 min). For these set of experiment we only consider *DynAGreenLife* algorithm as it balances system lifetime and system energy consumption. In Table 10, we compare the system lifetime given by our *DynAGreenLife* algorithm to *ILPLife-Far*, *ILPLife-Near* and *All-On-BE*. *ILPLife-Far* is the assignment obtained by *ILPLife* assuming the user is away from the base station and *ILPLife-Near* is the assignment computed by *ILPLife* assuming the user is closer to the base station. Improvements due to the dynamic adaptability of our algorithms depend on how different the system parameters are from the initial condition used for *ILP* solution. These two variations are used to cover *ILP* solution with the best and the worst communication costs between the cell-phone and base station.

We monitor link conditions every 1 minute (which is the LCM of all task periods) and run *DynAGreenLife* when the change in communication costs is 30% higher than the last run of the algorithm. We observe that while a person is moving in a car we can obtain up to 68% improvement system lifetime and 25% on an average compared to *ILPLife-Far* and up to 18% improvement compared to *ILP-Near*. If a person is on a bike or walking, we observe higher improvements compared to *ILPLife-Far*. A sample user location trace for these experiments indicates that the user remains closer to the base station most of the time and hence improvement over *ILPLife-Near* is not as significant as the improvement of *ILPLife-Far*.

6.1.6 Utilizing Multiple ILP Solutions

Looking at results in Table 10, one would suggest a simple heuristic technique in which we switch between task assignments provided by *ILPLife-Near* and *ILPLife-Far* depending on the user's proximity to a base station. Such a solution, labeled as *ILP-Flipflop* in Figure 18, would not handle other additional changes in the system such as changes in the workload, or in channel conditions between. To demonstrate this, we ran experiment with the same mobility described in the previous section, but with the addition of change in execution time of processing tasks every hour. Change in the execution time represents a practical scenario in which a processing task might be asked to produce better quality results by doing more processing. Figure 18 shows the results of our experiments for all three task sets. Our *DynAGreenLife* algorithm is up to 30% and on an average

21% better system lifetime than *ILP-Flipflop* technique described above. While the user is walking, the improvement is higher. The reason is that the communication costs lay between the best and the worst conditions longer as the user is moving at a slower speed.

7 Conclusion

Task assignment in a mobile sensing system consisting of heterogeneous resources may significantly impact the lifetime and total energy consumption of the system. This work presents a number of static and dynamic task assignment strategies to save energy and extend the system lifetime. Two Integer Linear Program (*ILP*) algorithms are formulated to generate a design time optimal task assignment to be used as a baseline for comparison. Given the dynamically changing nature of mobile systems and computationally expensive nature of *ILP*s, a heuristic algorithm (*DynALife*) and two dynamic graph-based partitioning algorithms (*DynAGreen* and *DynAGreenLife*) are presented in this paper. These three algorithms are computationally efficient and are able to adapt in real-time to the changing system conditions. All these algorithms are implemented in Qualnet to simulate various real-time system scenarios. Task assignments generated by our dynamic algorithms perform similar to the optimal task assignment of *ILP* in static conditions. Our algorithms have 1.4 times longer system lifetime compared to processing all data on backend server. They also outperform *ILP*-based static solutions by adapting at runtime to dynamically changing conditions. Results show that our techniques perform up to 88% better than the *ILP* based solutions. *DynAGreenLife* has up to 43% longer battery lifetime compared to *DynAGreen* algorithm and consumes up to 40% less energy than *DynALife* algorithm. Thus, *DynAGreenLife* algorithm balances both system lifetime and energy.

ACKNOWLEDGMENTS

This work has been funded by NSF project, “Citisense” grant CNS-0932403, UCSD’s Center for Networked Systems and Qualcomm. This work was also supported in part by the TerraSwarm Research Center, one of six centers supported by the STARnet phase of the Focus Center Research Program (FCRP) a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

References

[1] S.C.K. Lam, K. L. Wong, K. O. Wong, W. Wong, W. H. Mow, “A smartphone-centric platform for personal health monitoring using wireless wearable biosensors”, Information, Communications and Signal Processing Conference, IEEE, pp.1-7, 2009

[2] C. Palazzi, D. Maggiorini, “From playgrounds to smartphones: Mobile evolution of a kids game”, Consumer Communications and Networking Conference (CCNC), IEEE , pp.182-186, 2011

[3] D. Roggen, C. Lombriser, M. Rossi, G. Tröster, “Titan: an enabling framework for activity-aware "pervasive apps" in opportunistic personal area networks”, opportunistic personal area networks, EURASIP Journal on Wireless Communications and Networking, vol. 2011, pp. 1-22 , 2011

[4] L. Schmid, T. Holleczek, and G. Tröster, “Ridersguide: The first real-time navigation system for ski slopes”, Eingebettete Systeme, Informatik aktuell, vol 1, Springer Berlin Heidelberg, pp. 71-80, 2011

[5] K. Sakamura ,N. Koshizuka, “Ubiquitous computing technologies for ubiquitous learning”, Wireless and Mobile Technologies in Education, IEEE International Workshop on, pp.11-20, 2005

[6] P. Honeine, F. Mourad, M. Kallas, H. Snoussi, H. Amoud, C. Francis, “Wireless sensor networks in biomedical: Body area networks”, Systems, Signal Processing and their Applications (WOSSPA), 7th International Workshop on, pp. 388-391, 2011

[7] A. Salarian, H. Russmann, F. Vingerhoets, P. Burkhard, K. Aminian, “Ambulatory monitoring of physical activities in patients with parkinson’s disease”, IEEE Transactions on Biomedical Engineering, vol.54, no. 12, pp. 2296-2299, 2007

[8] H. L. Chan, P. K. Chao, Y.C. Chen, W. J. Kao, “Wireless body area network for physical-activity classification and fall detection”, Medical Devices and Biosensors, 5th International Summer School, pp. 157-160, 2008

[9] E. Kantoch, M. Smolen, P. Augustyniak, P. Kowalski, “Wireless body area network system based on ecg and accelerometer pattern”, Computing in Cardiology, pp. 245-248, 2011

[10] C. Otto, E. Jovanov, A. Milenkovic, “A WBAN-based system for health monitoring at home”, Medical Devices and Biosensors, 2006 3rd IEEE/EMBS International Summer School, pp. 20-23, 2006

[11] S. Munir, B. Ren, W. Jiao, B. Wang, D. Xie, M. Ma, “Mobile wireless sensor network: Architecture and enabling technologies for ubiquitous computing”, Advanced Information Networking and Applications Workshops, 21st International Conference on, vol.2, pp. 113-120, 2007

[12] G. Yovanof, G. Hazapis, “An architectural framework and enabling wireless technologies for digital cities and intelligent urban environments”, Wireless Personal Communications: An International Journal, vol. 49, Issue 3, pp. 445-463, 2009

[13] Z. Chaczko, F. Ahmad, V. Mahadevarr, “Wireless sensors in network based collaborative environments”, Information Technology Based Higher Education and Training, 2005

[14] E. Jeannot and B. Knutsson, “Adaptive online data compression”, HPDC-11, pp. 379-388, 2002

[15] R. Xu, Z. Li, C. Wang, and P. Ni, “Impact of data compression on energy consumption of wireless-networked handheld devices”, International Conference on Distributed Computing Systems, IEEE, 2003

[16] K. Wac, R. Bults, B. van Beijnum, I. Widya, V. Jones, D. Konstantas, M. Vollenbroek-Hutten, H. Hermens, “Mobile Patient Monitoring: the MobiHealth System”,' Engineering in Medicine and Biology Society(EMBS), IEEE, pp. 1238-1241, 2009

[17] F. Dabiri, A. Vahdatpour, H. Noshadi, H. Hagopian, M.Sarrafzadeh “Ubiquitous personal assistive system for neuropathy”, Health Care and Assisted Living Environments(HealthNet), ACM, no. 17, 2008

[18] A. Wood, G. Virone, T. Doan, Q. Cao, L. Selavo, Y. Wu, L. Fang, Z. He, S. Lin, J. Stankovic, “Alarm-net: Wireless sensor networks for assisted-living and residential monitoring”, Technical. Report CS-2006-13, University of Virginia, 2006

[19] W. Wu, L. Au, B. Jordan, T. Stathopoulos, M. Batalin, W. Kaiser, A. Vahdatpour, M. Sarrafzadeh, M. Fang, J. Chodosh, “The SmartCane system: an assistive device for geriatrics”, Body area networks(BodyNets), ICST 3rd International conference on, no.2, pp. 1-4, 2008

[20] Q. Wang, W. Shin, X. Liu, Z. Zeng, C. Oh, B. AlShebli, M. Caccamo, C. Gunter, E. Gunter, J. Hou, K. Karahalios, L. Sha, “I-living: An open system architecture for assisted living”, Systems Man and Cybernetics(SMC), IEEE, 2006

[21] H. Aghajan, J. C. Augusto, P. Mccullagh, J. ann Walkden, “Distributed vision-based accident management for assisted living”, International Conference on Smart homes and health telematics(ICOST), Springer-Verlag, pp. 196-205, 2007

[22] Y. Tian, E. Ekici, F. Ozguner, “Energy-constrained task mapping and scheduling in wireless sensor networks”, Mobile Adhoc and Sensor Systems(MASS) IEEE, pp. 211-218, 2005,

[23] Y. Tian, E. Ekici, “Cross-layer collaborative in-network processing in multihop wireless sensor networks”, IEEE Transactions on Mobile Computing, vol. 6, no. 3, pp. 297-310, 2007

[24] C.U. Subrahmanya, B. Veeravalli, Y. Liu, S. Viswanathan, “On the design of static and dynamic energy-aware task mapping algorithms for body area networks”, Medical Devices and Biosensors(ISSS-MDBS), International Summer School and Symposium, pp. 156-156, 2008

[25] J. K. Lenstra, D. B. Shmoys, E. Tardos, “Approximation algorithms for scheduling unrelated parallel machines”, Journal Mathematical Programming: Series A and B, vol. 46, no. 3, pp. 259-271, 1990

[26] S. Bokhari, “Partitioning problems in parallel, pipeline, and distributed computing”, IEEE Transactions on Computers, vol. 37, no. 1, pp. 48-57, 1988

[27] H. Stone, “Multiprocessor scheduling with the aid of network flow algorithms”, IEEE Transactions on Software Engineering, vol. SE-3, no. 1, pp. 85-93, 1977

[28] A. Pathak, V. Prasanna, “Energy-efficient task mapping for data-driven sensor network macroprogramming”, IEEE Transactions on Computers, vol. 59, no. 7, pp. 955-968, 2010

[29] “LP-solve.” http://lpsolve.sourceforge.net/5.5/

[30] C. Lombriser, D. Roggen, M. Stäger, G. Tröster, “Titan: A tiny task network for dynamically reconfigurable heterogeneous sensor networks”, Kommunikation in Verteilten Systemen(KiVS), Springer Berlin Heidelberg, pp. 127-138, 2007

[31] Y. Boykov, V. Kolmogorov, “An experimental comparison of mincut/max- flow algorithms for energy minimization in vision”, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2004

[32] Personal Activity and Location Measurement System (PALMS) project, http://ucsd-palms-project.wikispaces.com/

[33] B. Demchak,.J. Kerr, F. Raab, K. Patrick, I.H. Kruger, "PALMS: A Modern Coevolution of Community and Computing Using Policy Driven Development," Hawaii International Conference on System Sciences(HICSS), pp. 2735-2744, 2012

[34] Scalable Network Technologies, “Qualnet simulator,” http://www.qualnet.com

[35] Y. Yu, V. K. Prasanna., “Energy-balanced task allocation for collaborative processing in wireless sensor networks”, Journal Mobile Network Applications, vol. 10, no. 1-2, pp. 115-131, 2005

[36] P. Aghera, D. Fang, T. Simunic Rosing, K. Patrick, "Energy management in wireless healthcare systems", Information Processing in Sensor Networks(IPSN), IEEE, pp. 363-364, 2009

[37] P. Aghera, D. Krishnaswamy, D. Fang, A. Coskun, T. Simunic Rosing, "DynAHeal: Dynamic Energy Efficient Task Assignment for Wireless Healthcare Systems", Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1661-1664, 2010

[38] P. Aghera, D. Krishnaswamy, and T. S. Rosing, , "DynAGreen: Hierarchical Dynamic Energy Efficient Task Assignment for Wireless Healthcare Systems", Body area Networks(BodyNets), pp. 33-36, 2010

FIGURES AND TABLES

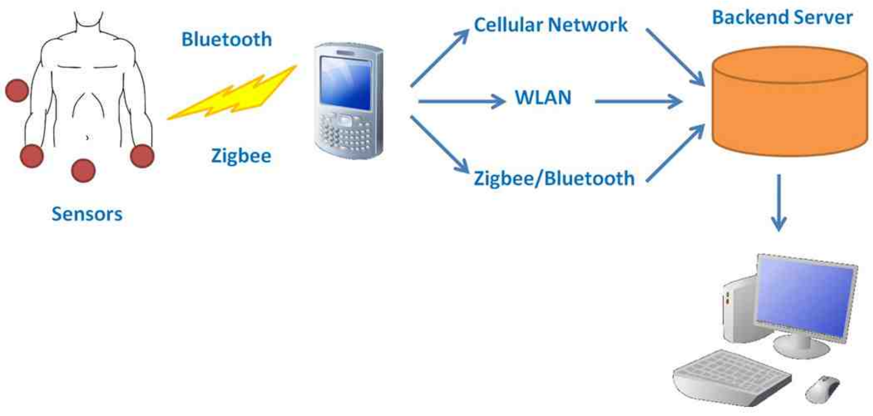


Figure 1. Wireless body area network architecture

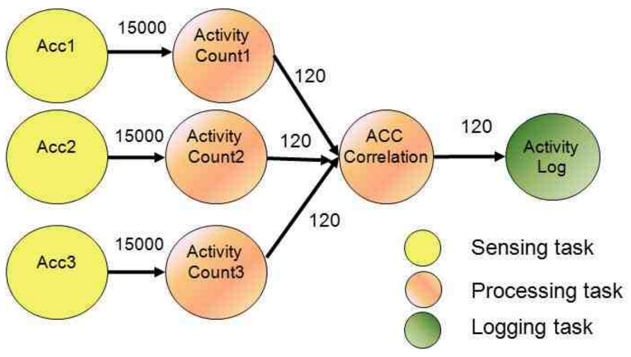


Figure 2. Example task graph

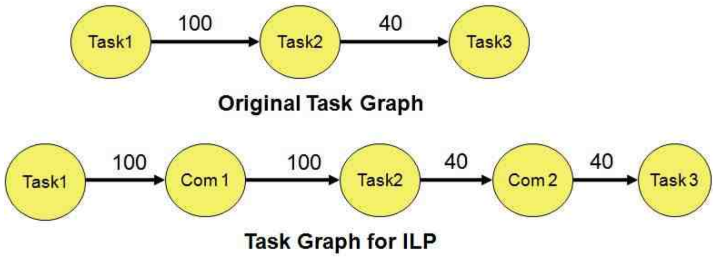


Figure 3. Task graph transformation for ILP

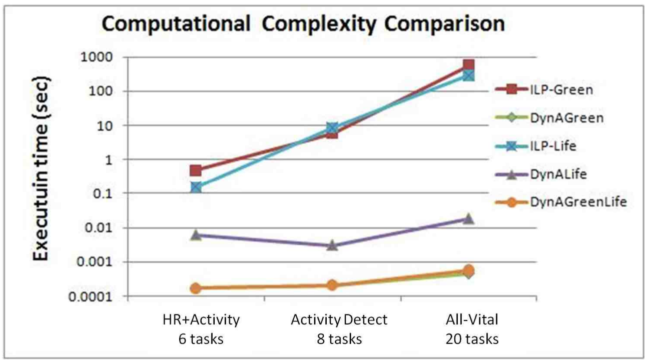


Figure 4. Computational overhead of *ILP* and dynamic algorithms running on 1.8GHz CPU

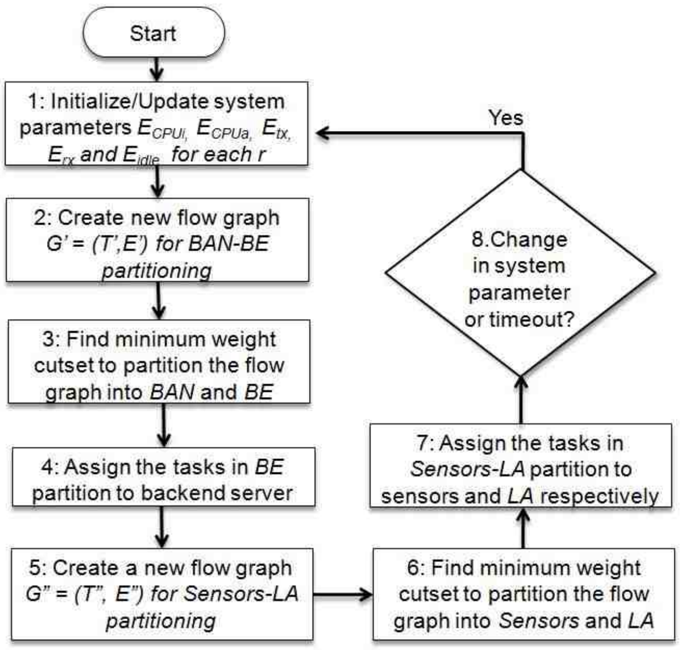

Start
1: Initialize/Update system parameters E_{CPUi}, E_{CPUa}, E_{tx}, E_{rx} and E_{idle} for each r
2: Create new flow graph G' = (T',E') for BAN-BE partitioning
3: Find minimum weight cutset to partition the flow graph into BAN and BE
4: Assign the tasks in BE partition to backend server
5: Create a new flow graph G" = (T", E") for Sensors-LA partitioning
6: Find minimum weight cutset to partition the flow graph into Sensors and LA
7: Assign the tasks in Sensors-LA partition to sensors and LA respectively
8.Change in system parameter or timeout?
Yes

Figure 5. DynAGreen Algorithm

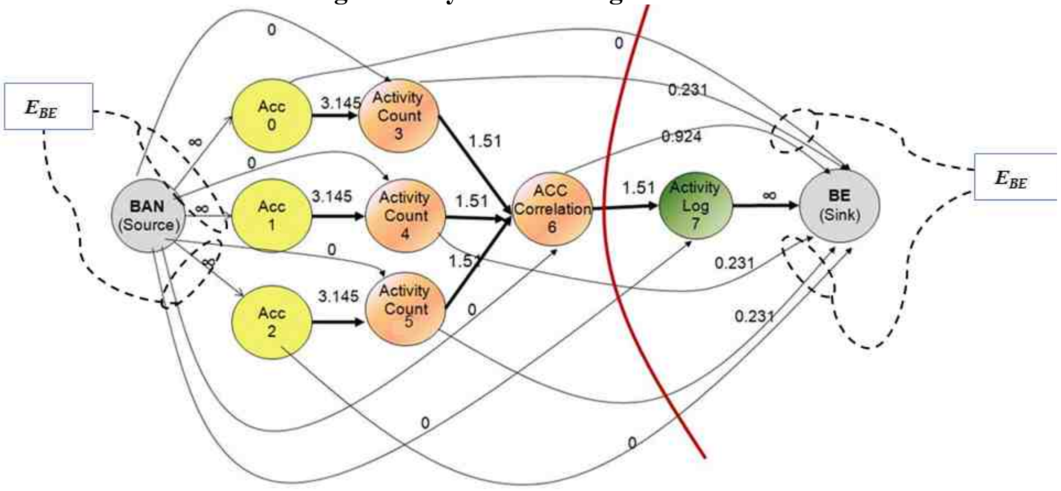


Figure 6. BAN-BE partition

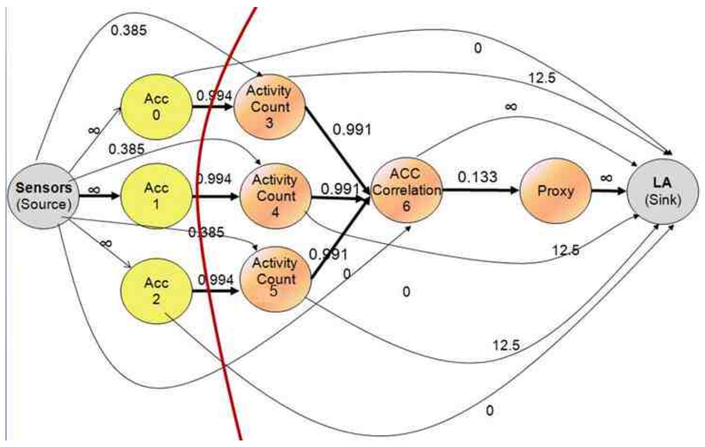


Figure 7. Sensor-LA partition

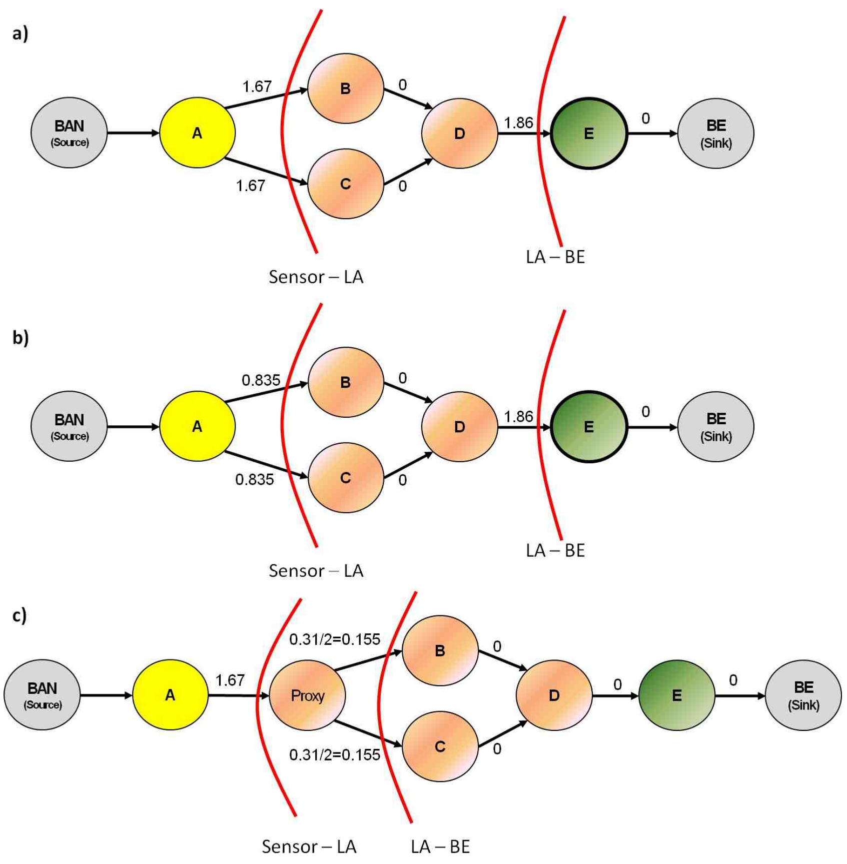


Figure 8. Task assignment for task set with task that have an out degree greater than 1. Only the communication cost is shown on the graph edges. a) Initial assignment. b) Reduction of the weight when the same data is sent to multiple tasks. c) Second assignment including a *Proxy* task that takes care to forward the data between the sensor node and the *BE*.

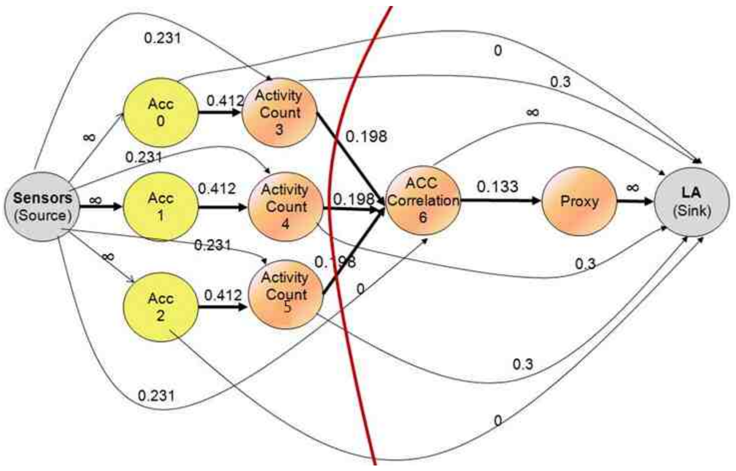
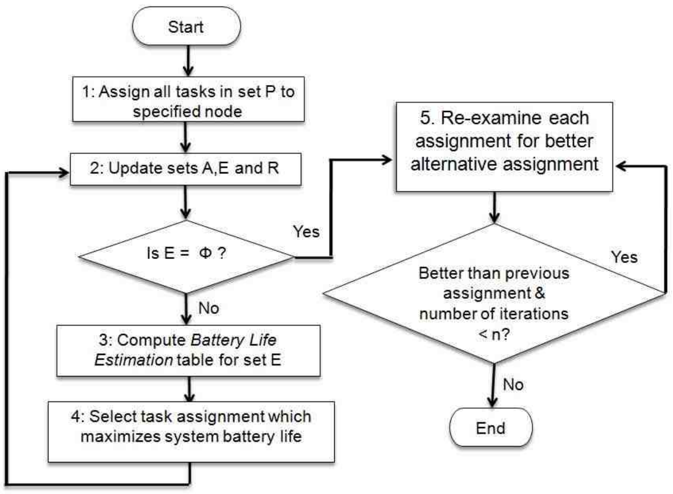


Figure 9. Sensor-LA partition



Start
1: Assign all tasks in set P to specified node
2: Update sets A,E and R
Is E = Φ ?
Yes
No
3: Compute *Battery Life Estimation* table for set E
4: Select task assignment which maximizes system battery life
5. Re-examine each assignment for better alternative assignment
Better than previous assignment & number of iterations < n?
Yes
No
End

Figure 10. DynALife Algorithm

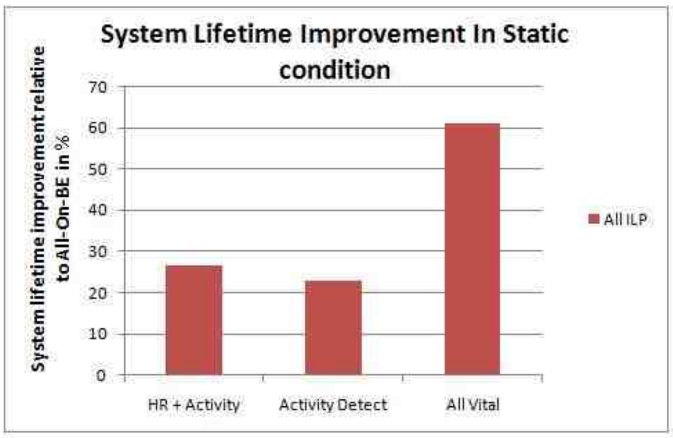


Figure 11. Percentage improvement in system battery lifetime achieved in static conditions compared to *All-On-BE* for both ILPs and for our dynamic algorithms.

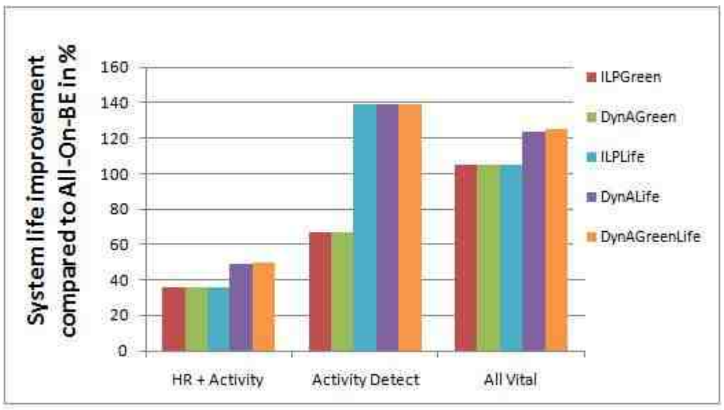


Figure 12. Percentage improvement in system battery lifetime compared to *All-On-BE*

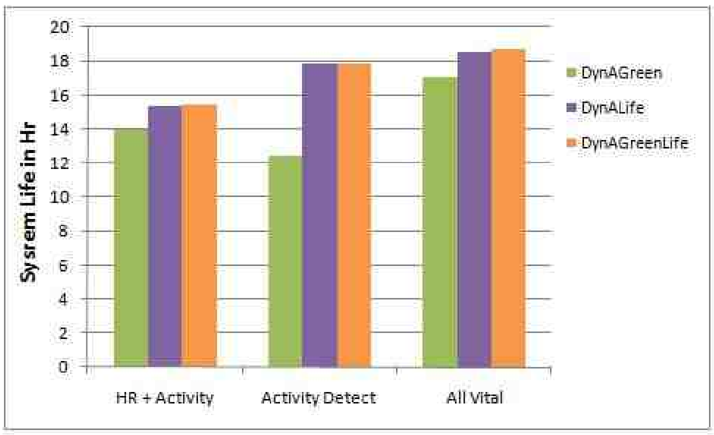


Figure 13. Comparison of system battery lifetime achieved by dynamic algorithms

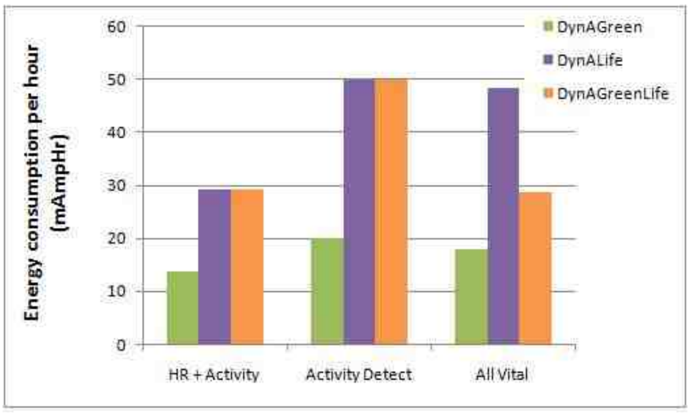


Figure 14. Comparison of system energy achieved by dynamic algorithms

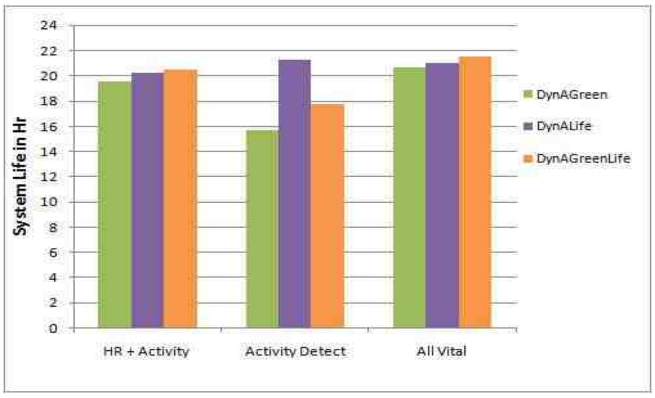


Figure 15. System lifetime comparison between dynamic algorithms with change in wireless channel conditions

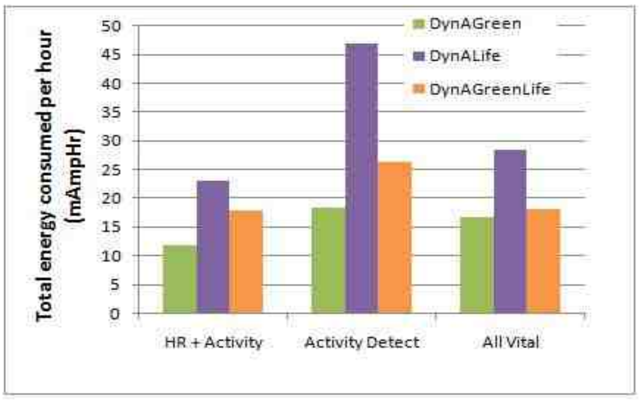


Figure 16. System energy consumption comparison with change in wireless channel conditions

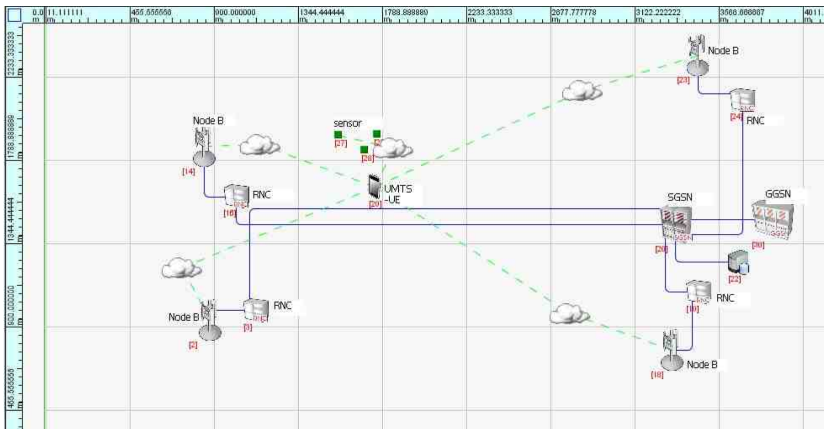


Figure 17. A screenshot of the simulation terrain. The *LA* (*UMTS-UE*) can communicate via cellular network to multiple base stations (*Node B*) which act as gateway to the *BE* (GGSN, *SGSN*). The body worn sensors (*sensor*) collect data from the user and send it to the *LA*. Note that multiple user can operate in parallel. Each network of body worn sensors communicates with their respective *LA*.

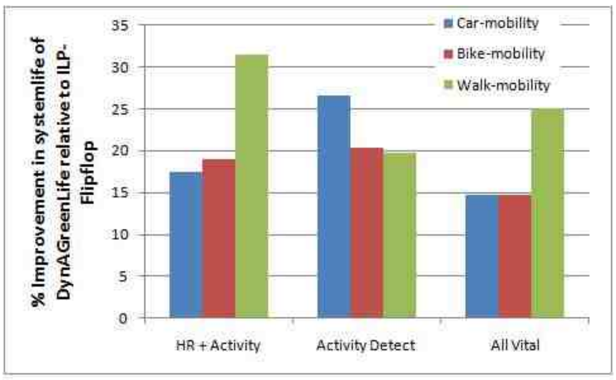


Figure 18. Percentage improvement in system lifetime achieved by *DynAGreen-Life* relative to *ILP-Flipflop* with changing conditions

Table 1

VARIABLES AND CONSTANTS

$\boldsymbol{E_{Br}}$	Battery capacity of resource r
$\boldsymbol{P_{Rr}}$	Power consumed by the receiver of resource r
$\boldsymbol{P_{Tr}}$	Power consumed by the transmitter of resource r
$\boldsymbol{P_{Er}}$	Power consumed in the execution of a task by resource r
$\boldsymbol{t_{ir}}$	Execution time of task i on resource r
$\boldsymbol{w_{ij}}$	Weight of the edge from task i to task j
$\boldsymbol{b_{ir}}$	$b_{ir} \in \{0,1\}$ s.t. $b_{ir} = 1$ if task i is assigned to resource r
$\boldsymbol{\mu_{Rr}}$	Average data rate of the receiver on resource r
$\boldsymbol{\mu_{Tr}}$	Average data rate of the transmitter on resource r
$\boldsymbol{pred_{ij}}$	$pred_{ij} \in \{0,1\}$ s.t. $pred_{ij} = 1$ iff task j is a predecessor of task i
$\boldsymbol{suc_{ij}}$	$suc_{ij} \in \{0,1\}$ s.t. $suc_{ij} = 1$ iff task j is a successor of task i
$\boldsymbol{E_{Rir}}$	Energy consumed by a resource r to receive input data for task i = $\sum_{j=1}^{n}\{pred_{ij} * (1-b_{jr}) * P_{Rr} * (w_{ji}/\mu_{Rr})\}$
$\boldsymbol{E_{Tir}}$	Energy consumed by a resource r to transmit input data for task i $=\sum_{j=1}^{n}\{succ_{ij} * (1-b_{jr}) * P_{Tr} * (w_{ij}/\mu_{Tr})\}$
$\boldsymbol{E_{r}}$	Total energy consumed by a resource r = $\sum_{j=1}^{n}\{b_{ij} * (E_{Rir} + P_{Er} * t_{ir} + E_{Tir})\}$
$\boldsymbol{s_{i}}$	Start time of task i
$\boldsymbol{\tau_{i}}$	End time of task i
$\boldsymbol{d_{i}}$	Deadline of task i

Table 2

***ILP* OBJECTIVE AND CONSTRAINTS**

***ILP*Green :=** $\min\left(\sum_{r=1}^{m-1} E_r\right)$	Minimize system energy consumption
***ILP*Life :=** $\min\left\{\max_r \frac{E_r}{E_{Br}}\right\} \quad where, r = 1, ..., m-1$	Maximize system lifetime
(a) $\forall i, \sum_{r=1}^{m} b_{i,r} = 1$	Total Allocation Constraint: Each task is assigned to one and only one resource
(b) $E_r \leq E_{Br}$	Battery Capacity Constraint
(c) $(b_{jm} + b_{jm-1}) = 1$ *if* $(b_{im} = 1 \ \wedge \ pred_{ij} = 1)$	Resource Allocation Constraint 1: Predecessor tasks of a task mapped to the backend are mapped to either the backend or the local aggregator
(d) $(b_{jm-1} + \sum_{k=1}^{m-2} b_{jk}) = 1$ *if* $(b_{im-1} = 1 \ \wedge \ pred_{ij} = 1)$	Resource Allocation Constraint 2: Predecessor tasks of a task mapped to the local aggregator are mapped to either the local aggregator or a sensor
(e) $b_{jk} = 1$ where $k=1,\ldots,m\text{-}2$ *if* $(b_{ik} = 1 \ \wedge \ pred_{ij} = 1)$	Resource Allocation Constraint 3: Predecessor tasks of a task mapped to a sensor are mapped to the sensor
(f) $s_i \geq \max(0, \tau_j) \forall j$ where $pred_{ij} = 1$	Start Time Constraint: The starting time for tasks must be greater than the finishing time of all its predecessor tasks
(g) $\tau_i = s_i + \sum_{r=1}^{n} \left\{ b_{ir} * (t_{ir} + \sum_{j=1}^{n} pred_{ij} * (1 - b_{jr}) + suc_{ij} * (1 - b_{jr})) \right\}$	Finishing Time Constraint: The finishing time of a task is the start time plus the time it takes to receive, execute, and send the data for the task
(h) $\tau_i \leq d_i$ For each sink task i in the task graph	Deadline Constraint

Table 3

WEIGHTS OF EDGES IN BAN-BE PARTITION

Edge	Value	Condition
***w*(*BAN*,*t*)**	0 ∞	If *t* is not bound to *BAN* If *t* is bound to *BAN*
***w*(*t*, *BE*)**	E_{CPU}(t, *LA*) 0 ∞	If *t* is not bound If *t* is bound to sensor or LA If *t* is bound to BE
$w_{wan}(t_p, t_s)$	$E_{radio}(t_p, LA)$	where radio = *LA*’s WAN radio

Table 4

WEIGHTS OF EDGES IN SENSOR-LA PARTITION

Edge	Value	Condition
w(Sensor,t)	$E_{CPU}(t, LA)$ 0 ∞	If task t is not bound If task *t* is bound to *LA* If task *t* is bound to a sensor
w(t, LA)	$E_{CPU}(\mathrm{t}, Sensor)$ 0 ∞	If task *t* is not bound If task *t* is bound to a sensor If task *t* is bound to *LA*
$\boldsymbol{w_{wan}(t_p, t_s)}$	$E_{radio}(t_p, Sensor)$	where radio = source sensor's radio

Table 5

BATTERY LIFETIME ESTIMATION TABLE

Task ID	Sensor lifetime if on (sensor, LA, BE) *sec*	LA lifetime if on (sensor, LA, BE) *sec*	System lifetime if on (sensor, LA, BE) *sec*	Deadline slack if on (sensor, LA, BE) *msec*	Max system lifetime *sec*	Best assign. node ID
1	(600,1200,1020)	(800,1500,1500)	(600,1200,10200	(500,350,200)	(1200)	LA

Table 6

SENSOR NODE AND LA'S POWER CONSUMPTION

	Sensor node (mW)	***LA (mW)***
First Allocation	2.77	1.70
Second Allocation	2.77	1.07
% Savings	0.0	38.8

Table 7

WORKLOADS

Taskgraph	# of sensors	# of tasks	Application
HR + 1 accelerometer activity detection	2	6	ECG sensor detects heart rate per minute & accelerometer keeps track of activity.
3 accelerometers activity detection	3	8	Detects a person's activity using three accelerometers.
All-Vital	5	20	Log all vital signs like heart rate, blood pressure, activity in addition to individual's location.

Table 8

PERCENTAGE IMPROVEMENT IN SYSTEM LIFETIME WITH VARYING WIRELESS CHANNEL CONDITIONS

Algorithm	*Taskgraph*	***% System lifetime improvement achieved by DynAGreenLife relative to***		
		ILP-Green	*ILP-Life-Far*	*ILP-Life-Near*
DynAGreenLife	% HR + Activity	23.79	23.79	66.68
	% Activity Detection	24.88	-6.53	37.47
	% All-Vital	22.14	22.14	4.28

Table 9

PERCENTAGE IMPROVEMENT IN SYSTEM LIFETIME WITH DYNAMIC LOAD CHANGES

Algorithm	*Taskgraph*	***% System lifetime improvement achieved by DynAGreenLife relative to***		
		ILP-Green	*ILP-Life-Far*	*ILP-Life-Near*
DynAGreenLife	% HR + Activity	20.51	20.51	6.12
	% Activity Detection	40.86	14.66	14.66
	% All-Vital	8.74	8.74	17.02

Table 10

PERCENTAGE IMPROVEMENT IN SYSTEM LIFETIME OF DynAGreenLife WITH RANDOM MOBILITY

Task graph: HR + Activity				
Speed	Pause duration	**% Improvement in battery lifetime relative to**		
		ILP-Far	*ILP-Near*	*All-On-BE*
Car	15 min	24.17	3.15	47.19
Car	45 min	5.90	3.39	18.68
Bike	15 min	15.87	6.17	26.84
Bike	45 min	8.36	8.36	18.70
Walk	15 min	9.58	3.62	19.67
Walk	45 min	23.42	5.78	47.46

Task graph: Activity Detection				
Speed	Pause duration	**% Improvement in battery lifetime relative to**		
		ILP-Far	*ILP-Near*	*All-On-BE*
Car	15 min	67.81	17.23	67.81
Car	45 min	47.88	6.84	47.88
Bike	15 min	34.53	9.71	34.53
Bike	45 min	87.98	5.7	87.98
Walk	15 min	43.15	6.12	43.15
Walk	45 min	32.51	12.09	32.51

Task graph: All Vital				
Speed	Pause duration	**% Improvement in battery lifetime relative to**		
		ILP-Far	*ILP-Near*	*All-On-BE*
Car	15 min	21.36	2.3	70.07
Car	45 min	12.06	3.9	42.34
Bike	15 min	22.44	2.45	45.93
Bike	45 min	8.64	3.48	42.8
Walk	15 min	14.18	8.23	46.73

Walk	45 min	34.88	7.36	56.59

BIOGRAPHIES

Priti Aghera obtained her M.S. degree in Computer Science from University of California, San Diego. Her thesis topic was Energy Measurement in Wireless Healthcare Systems Using Dynamic Task Assignment. Currently she is working as staff engineer II in Broacomm's Bluetooth group. Her focus at Broadcomm is Health Device Profiles and Bluetooth Low Energy.

Jinseok Yang received his M.S. degree in Electrical Engineering from the University of California, San Diego in 2011. Since 2012 he has been pursuing his PhD at UCSD within the System Energy Efficiency Lab. His reaearch is on wireless sensor networks and embedded systems.

Piero Zappi received Ph.D. and M.S. degrees in Electronics from the University of Bologna in 2009 and 2005 respectively. From 2010 to 2012 he held a post-doctoral position at University of California San Diego (UCSD) within the System Energy Efficiency Lab (SEE Lab). He is currently working at Qualcomm on projects related to energy efficiency and context awareness.

Dilip Krishnaswamy received a PhD in Electrical Engineering from the University of Illinois at Urbana-Champaign in 1997. He was a platform architect at Intel. He is a Sr. Staff Researcher at the Qualcomm Research Center in San Diego. He serves as Associate Editor-in-Chief of the IEEE Wireless Communications magazine. His current research interests are in wireless distributed processing, distributed machine learning, m2m services and technologies, edge services, and nanobiological networks

Ayse K. Coskun is an assistant professor in the Electrical and Computer Engineering Department at Boston University. She received her MS and PhD degrees in Computer Science and Engineering from University of California, San Diego. Coskun's research interests are temperature and energy management, 3D stack architectures, computer architecture, and embedded systems. Prof. Coskun has worked at Sun Microsystems (now Oracle), San Diego prior to her current position at BU. She has served as an associate editor for ACM Transactions on Design Automation of Electronic Systems and IEEE Embedded Systems Letters.

Tajana Šimunić Rosing is currently an Associate Professor of Computer Science and Adjunct Associate Professor in the Electrical and Computer Engineering Department at UCSD. She is currently heading the effort in SmartCities as a part of DARPA and industry funded TerraSwarm center. Prior to that she led the energy efficient datacenters theme as a part of the MuSyC center. Her research interests are energy efficient computing, embedded and wireless systems. Prior to this she was a full time researcher at HP Labs while being leading research part-time at Stanford University. She finished her PhD in 2001 at Stanford University, concurrently with finishing her Masters in Engineering Management. Her PhD topic was Dynamic Management of Power Consumption. Prior to pursuing the PhD, she worked as a Senior Design Engineer at Altera Corporation.