UNIVERSITY OF CALIFORNIA, SAN DIEGO

Energy Management in Wireless Healthcare Systems Using Dynamic Task Assignment

A thesis submitted in partial satisfaction of the requirements for the degree
Master of Science

in

Computer Science

by

Priti Aghera

2010

DEDICATION

I dedicate this thesis to my wonderful Mom. She have always been supportive and made countless sacrifices for me. Both this thesis and the completion of my graduate degree would not have been feasible without her constant motivation. Despite the miles that separate us I felt she was close to me and that gave me strength.

Love you Mom !!!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Tajana Simunic Rosing tajana for encouraging me to explore new research area. Her guidance has been invaluable through all stages of this thesis and my graduate degree. I must also thank my doctoral committee member, Dr. Dilip Krishnaswamy for his continuous feedback and contributions. Dilip was always available for brain storming whenever I was stuck at any point during the course of my research work. I appreciate Dr. Kevin Patrick and Fred Raab from UCSD school of medicine for providing me an opportunity to work on the latest projects on wireless healthcare.

I cannot thank my husband Parixit Aghera enough for his love, support and patience during stressful times. I am thankful for his contributions to my academic career since my early years in USA, and for his insightful advice during critical times. The greatest thanks are for all of my family, who had my best interest in their hearts at all times, who loved, supported and motivated me. I feel extremely lucky to have such a big and wonderful family. Especially my father-in-law who has always encouraged my curiosity and my ambition to follow my dreams. My parents have made me the person I am today, taught me to believe in myself, and reminded me what is important in life.

I would like to thank all my lab mates, co-authors, and friends at UC San Diego, especially Gaurav Dhiman, Raid Ayoub, Shervin Sharifi, Diana Fang, Ayse Coskun, Giacomo Marchetti, Edoardo Regini, Nima Nikzad, Bryan Kim, Jamie Steck, Piero Zappi, Dennis Dondi for their friendship and contributions to my research during my MS years. I have been surrounded with an exceptional circle of friends, and would like to thank especially Chetan Malvania, Jay Parikh, Abhishek Bhandari, Jesal Patel, Anshuman Gupta and Sravanthi for their support, tolerance, and for sharing the ups and the downs.

The research that forms the basis of this dissertation has been funded in parts by NSF Citisense grant CNS-0932403, NSF grant 0729029, Center for networked systems (http://cns.ucsd.edu) grant CNS08TR and Qualcomm.

PUBLICATIONS

P. Aghera, D. Krishnaswamy,T.Rosing, ”DynAGreen: Hierarchical Dynamic Energy Efficient Task Assignment for Wireless Healthcare Systems”, BODYNETS 2010

P. Aghera, D. Krishnaswamy, D. Fang, A. Coskun, T. Rosing, ”DynAHeal: Dynamic Energy Efficient Task Assignment for Wireless Healthcare Systems”, DATE 2010.

P. Aghera, D. Fang, T. Simunic Rosing, K. Patrick, ”Poster Abstarct: Energy management in wireless healthcare systems”, IPSN 09.

ABSTRACT OF THE THESIS

Energy Management in Wireless Healthcare Systems
Using Dynamic Task Assignment

by

Priti Aghera

Master of Science in Computer Science

University of California, San Diego, 2010

Tajana Simunic Rosing, Chair

Wireless healthcare systems are hierarchical and heterogeneous in nature with components that have different energy and performance capabilities. Ensuring the optimal energy consumption across all these components while meeting performance requirements is a critical issue. In such systems with processing, sensing, and communication tasks, allocation of tasks to devices of the system affects the system battery lifetime and energy consumption.

This thesis presents a number of static and dynamic task assignment strategies to save energy and extend system lifetime. The problem of optimal task assignment with objectives related to minimizing the total energy consumption and maximizing the system lifetime are formulated using *Integer Linear Program (ILP)*-based solutions. The *ILP* based solutions are able to improve the battery lifetime by up to 1.4 times compared to performing all of the processing tasks on the backend server.

Given the dynamic nature of wireless systems, three dynamic algorithms are proposed. These algorithms are computationally efficient and are able to adapt to changing system conditions in real-time unlike *ILP* based solutions. *DynAGreen* algorithm is a graph-based task assignment algorithm with the objective of minimizing total system energy consumption. *DynALife* algorithm is a heuristic task assignment strategy that extends system battery lifetime. *DynAGreenLife* balances both system energy and system lifetime in wireless healthcare systems. Our dynamic scheduling techniques are able to improve system lifetime by up to 88% and on an average 30% in comparison to the static task assignment given by the *ILP* in dynamically changing urban conditions that represent real life scenarios.

Chapter 1

Introduction

The new generation of wireless mobile systems with seamless integration of 2.5G and 3G cellular systems, wireless LAN, Bluetooth, and Zigbee can provide wide coverage and an improved capacity to run many different types of wireless applications. Advances in integrated circuit design and bioengineering have led to the design of low-cost, miniature, lightweight, physiological sensors that can be seamlessly integrated into a body area networks for human health monitoring.

Wireless sensor-based healthcare systems are hierarchical and heterogeneous in nature. They consist of different types of components with various energy capacities, processing capabilities, and wireless communication capabilities. These components include multiple sensors, one or more local data aggregators (e.g. cell phones), and a server. The primary goal of today's wireless healthcare systems is to continuously sense various vital signs of a subject and monitor environmental parameters such as temperature, carbon monoxide (CO), pressure, process these raw signals and send the information to a backend server for analysis by healthcare professionals in real time using wireless communication channels. One or more intermediate nodes, called local aggregators in this work, can receive information from the sensors and then forward such information to a remote server node.

There is a recent push to provide real time feedback to the users of wireless

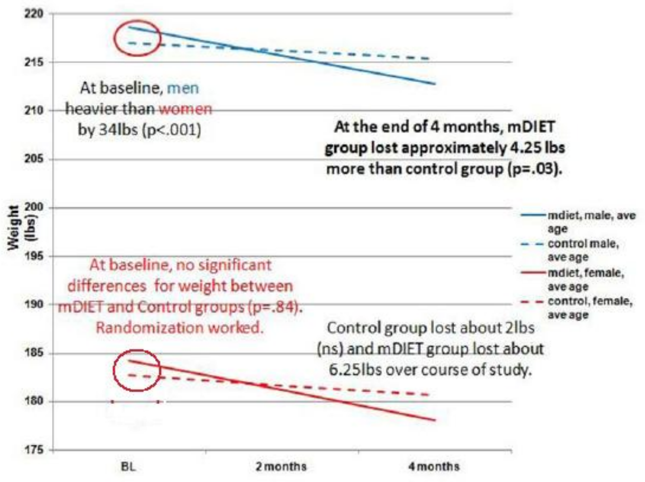

Figure 1.1: mDiet: Clinical trial results

healthcare systems with the goal of changing their behaviour. This in tern requires better CPUs, relatively, power, etc. We assume that a local aggregator has Wide Area Wireless Access Network (WWAN) connectivity using a 3G/4G wireless protocol to reach the server providing service continuity with patient mobility.

An example of such a system is the wireless preventive healthcare project run at UCSD. This project focuses on encouraging study subjects to increase their physical activity and optimize their energy intake through real time monitoring and feedback to the study subjects via sensors (e.g. accelerometer, GPS, heart rate) and cell phones. During a 16 week clinical trial [10], the group using this system lost an average of 4.4 pounds more than the control group as shown in Fig. 1.1. More than 96% of subjects loved the system over the traditional paper based survey and would recommend it to others. The study showed that real-time feedback to users is much more successful in encouraging change in human behaviour.

One of the key challenges encountered during this clinical trials is managing energy consumption of the mobile devices used in the system due to limited battery lifetime. The cost of computing locally is lower than wirelessly transmitting data to a remote node ([9] [22]). Thus, in addition to communicating data to the server, we can leverage the processing capability of today's mobile phones and sensors to perform some processing of the sensed data locally. Such processing may also be desirable if connectivity is weak or not available, so that alerts can be provided to the patient wherever they are, and regardless of the nature of the connectivity to the server. As this local processing consumes battery energy, it leads to a dynamic trade-off in processing and communications costs in the distributed system. Depending on the available processing capability, the available battery-life and the changing wireless conditions the task assignment on different nodes in this distributed system should vary.

1.1 Thesis Contribution

Our goal in this work is to develop an energy efficient assignment algorithm that assigns tasks to the resources with different energy optimizing objectives while meeting task dependency and communication constraints. We approach this problem of task assignment in following two ways:

1. Finding an optimal task assignment for any given system utilizing knowledge of all system parameters. Note that this solution is only optimal for given fixed set of system parameters.

2. Designing a lightweight and dynamic solution behaving close to an optimal solution but can adapt to changing system characteristics and changing taskset quickly.

With both the approaches we address following three different system objectives:

1. **Minimize system energy:** System energy is defined as a total of energy consumed by all mobile components(i.e. sensors and LA) in the system. Minimizing system energy can result in significantly different battery life of heterogeneous devices of such a system due to uneven workload distribution. Since a system lifetime is minimum of battery life of all devices, this objective, while fair across all nodes, would result in a node with low battery life being depleted of its battery energy faster and thus very short system lifetime. In a healthcare system

scenario where longer system lifetime is more important then minimizing overall energy of the system, objective# 2 should be used.

2. **Maximize system lifetime:** System lifetime is defined as the time from the start of the system until the time first device runs out of energy. Due to heterogeneity of the system, each device in the system can have very different battery capacity, thus leading to significantly different task allocation then objective# 1.

3. **Balance system energy and system lifetime:** This objective finds the middle ground, thus balancing the trade off between minimizing system energy and maximizing system lifetime.

To meet the objective of finding energy efficient task assignment, we formulate two different *ILP*s (Integer Linear Programs). *ILPGreen*, and *ILPLife* addresses the objective 1 and 2 respectively. While for a small number of tasks and resources the problem can be solved in polynomial time, for more complex systems the computational complexity of the *ILP* is too high. Additionally, dynamically changing system characteristics in wireless systems can invalidate the optimality of the statically computed task assignment. To address this, we designed three different dynamic algorithms which are computationally efficient and based on our simulations produces near optimal solutions. The algorithms *DynAGreen*, *DynALife* and *DynAGreenLife* address our system objectives 1, 2 and 3 respectively.

We show that in the static case, the optimal assignment of tasks to heterogeneous nodes in the system significantly affects performance and battery life of the system. In our experiments, we observed that the *ILP* solution is able to improve the battery lifetime by up to 1.4 times in comparison to performing all processing tasks on the backend server. We implement the proposed algorithms using Qualnet [15] discrete event wireless simulator. Our dynamic scheduling technique is able to improve system lifetime by up to 88% and on an average 30% in comparison to the static task assignment given by the *ILP* in dynamically changing urban conditions that represent real life scenarios. *DynAGreenLife* balances both system lifetime and system energy by extending system lifetime close enough to that given by *DynALife* but saves up to 40% more energy than *DynALife*. On the other hand *DynAGreenLife* extends system lifetime up to 40% more compared to *DynAGreen*.

The rest of the thesis is organized as follows. Chapter 2 discusses the prior work and Chapter 3 provides a formal description of the system model and task assignment problem. ILP based static optimal techniques for task assignment in wireless healthcare systems are described in Chapter 4. Chapters 5, 6 and 7 describes our proposed dynamic task assignment algorithms. Finally we present our results in Chapter 8.

Chapter 2

Related Work

Wireless health care applications can be categorized into i) disease management ii) assisted living for elderly and iii) preventive medicine. CardioNet [6] is one of the disease management applications, and it provides 24x7 cardiac monitoring service with beat-to-beat, real time analysis, automatic arrhythmia detection and wireless ECG transmission. The Mobihealth [3] project in Europe is another other example of disease management system. Dabiri et.al. presents the Electronic Orthotic Shoesystem as a means of ulcer prevention for patients suffering from neuropathy in [8]. There are many systems like Alarm-Net [20], SmartCane [21], I-Living [19], and PAMM [1] to aid the elderly in their day to day life. Similarly, [19] is an example of a wireless preventive healthcare system.

Most of such systems today assumes sense-and-forward mode where the data is gathered and sent to the back end server for processing. As we show later, this sense-and-forward mode does not leverage the processing capabilities of todays sensors and mobile phones and can lead to significantly lower system battery. Previous work has shown (e.g. [9], [22]) that transmitting or receiving a bit wirelessly can be significantly more expensive than processing the bit locally on the CPU. The processing energy of a task also depends on the processing power of the resource that task is computed. Thus the decision of processing locally or transmitting data for remote processing for a task depends on the processing power and battery of the resource where the task is assigned and number of bytes to be communicated.

Task assignment and scheduling onto multiple resources are classical problems in traditional computation and parallel computing [4] [16]. Stone [16] 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 tasks in the system. In our research, we assign tasks to different nodes in the system by minimizing system energy, and balancing the tradeoff between system energy and system lifetime. We utilize graph partitioning in a hierarchical manner to address partitioning between multiple nodes (server, local aggregator, and sensors) as we will describe later.

There is quite a bit of previous work on energy optimization in wireless sensor networks using efficient task assignment. Localized task mapping and task scheduling have been considered for homogeneous wireless sensor networks(WSN) [18], [23] and [17] recently. 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 [18]. Unlike our work, EcoMapS aims to schedule tasks with the minimum schedule length subject to energy consumption constraints. In the case of sensor failures, a quick recovery algorithm is executed to generate alternative schedule. This technique cannot be applied to our system as wireless healthcare systems have heterogeneous resources and each sensor has specific functionality. Energy-balanced task allocation of real time application onto single-hop cluster of homogeneous sensor nodes connected with multiple channels is introduced in [23] . Yu and Prasanna examine the two problems of assignment of jobs to resources and the determining of the voltage levels in a joint manner. They formulate the problem as an integer linear program (ILP), and they utilize a 3-phase heuristic to solve the problem. MTMS algorithm is proposed in [17] for task mapping and scheduling in multi-hop homogeneous WSNs with real time guarantees. Although task mapping and scheduling techniques has been applied to wired and wireless sensing networks very little has been done for heterogeneous and hierarchical wireless sensing systems. A.

Nahapetian et.al. has presented an overview of different power management techniques that can be very useful in tackling the power management issues facing wearable medical systems in [12]. The MicroLEAP platform [2] supports per-task real-time energy profiling to permit adaptive applications that select platform components to best match dynamically-varying measurement requirements.

Our work focuses on reducing the total energy consumption and maximizing lifetime of a wireless healthcare system by performing dynamic task assignment and scheduling. Contrary to most of the prior work in task assignment in WSN, our strategy is able to work with a heterogeneous set of resources with various performance and energy characteristics and a number of different types of connectivity. In addition, 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 resources.

Chapter 3

System Architecture and Model

This chapter describes the architecture of a typical wireless healthcare system in detail. Fig. 3.1 shows the architecture of a typical wireless distributed healthcare system. The system has three main components: Sensors, Local Aggregator (*LA*), and a Backend Server (*BE*). Sensors form the Body Area Network (*BAN*). Their function is to sense and detect specific events (e.g., low blood sugar). Sensors send processed and/or raw sample data to their respective *LA* wirelessly. There is one *LA* per *BAN*. The *LA* aggregates data collected by sensors and may process the data before sending it to a centralized *BE* for analysis by health professionalsor *LA* might send all the data to *BE* for analysis. A *LA* typically contains multiple wireless technologies such as Bluetooth, Zigbee, *WLAN*, and *WWAN*. The *BE* receives information sent by the *LA* and stores the information in a database which can be accessed for further analysis.

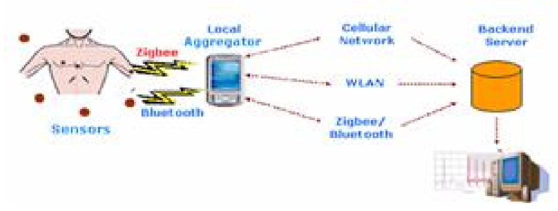

Figure 3.1: Wireless healthcare system architecture

Each resource has a CPU, local memory, and one or more communication units.

These devices differ in their processing capabilities, battery life, bandwidth, range, and types of radios. In a typical healthcare system, a patient would carry one *LA* (usually a cell phone). *BE* is a large computer server that contains a large database of all the information collected representing a computing cloud. Therefore, we assume that per patient there is only one *LA*, one *BE*, and a number of sensors. We also assume that data follows the path from sensors to *LA*, and then to *BE*. Sensors cannot communicate directly with *BE*, as they typically do not have the capability of transmitting data over a wide area network where the *BE* resides. Sensors in healthcare applications are usually designed for sensing and processing a specific task. They are not designed to handle other types of data. Thus, in our system, sensors only communicate with the *LA* and not with each other.

3.1 System Model

Tasks of a wireless healthcare application can be modelled as a Directed Acyclic Graph (*DAG*) $G = (T, E)$ where node set *T* represents set of *n* sensing/processing tasks, $T : i = 1, 2, ..n$ and *C* is set of edges that represent communication tasks between nodes, $C : i = 1, 2, ..n$. $C_{ij} \in C$ 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 for wireless healthcare systems 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 energy consumption is minimized or ii) the system lifetime is maximized. iii) objective i) and ii) are balanced.* Here system lifetime is defined as the minimum of battery life of all the mobile components per patient.

Table 3.1: Variables and Constants

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

Tasks of the DAG are further characterize by a set of varaiables and constants given in Table.3.1. 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.3.1 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 energy consumed by task i to receive its input data on resource r in terms of resource r's receive power P_{Rr}, its average receive datarate μ_{Rr} and time to receive task i's input data from its predecessor task j given by $(\frac{w_{ji}}{\mu_{Rr}})$. Similarly E_{Tir} defines the transmit energy associated with task i on resource r. A task i's computation energy is given by $P_{Er} * t_{ir}$. The total energy E_r consumed by a resource r, is the sum of computation energy of all tasks assigned on r, energy to receive input of these assigned tasks and transmission energy to transmit the output of these tasks to its successor on other resources.

Fig. 3.2 is an example of a *DAG* for activity detection application. 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 the 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,$suc_{ACC1,ActivityCount1} = 1$, $suc_{ACC2,ActivityCount2} = 1$, $suc_{ACC3,ActivityCount3} = 1$, $pred_{ActivityCount1,ACC1} = 1$, $pred_{ActivityCount2,ACC2} = 1$ 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 $s_{ActivityCount1} > tau_{ACC1}$.

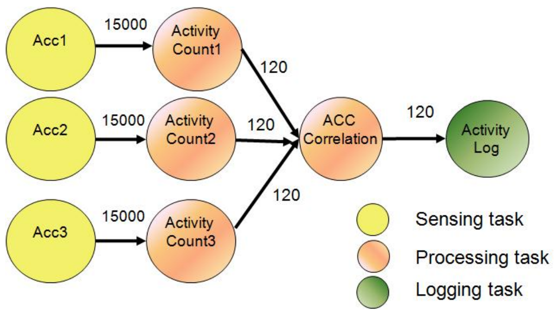

Figure 3.2: Wireless healthcare system architecture

Each resource of the system has battery capacity (defined as E_{Br} in Table.3.1. If cellphone has a battery energy of 300 mAmp then $E_{B_{LA}} = 300$. The total energy(E_r) consumed by a resource is the sum of energy consumed by the resource in receiving data from its predecessors (E_{Rir}), energy consumed in transmitting data to its successors (E_{Tir}) and execution time of tasks assigned to it($P_{Er} * t_{ir}$). Considering the processing capabilities of the resources for the case of the taskgraph shown in Fig. 3.2, we see that processing tasks can be done on either of the three tiers but with different performance costs. For example, if *Acc-Count1* is done on the *LA* or the *BE*, there is an added energy cost of communicating 15000 bytes from the *BAN* to the *LA* and then from the *LA* to the *BE*, compared to communicating only 120 bytes of processed data if the processing is done on the sensor itself. The time *LA* takes to transfer the data(15000 bytes) needed by the *Activity Count1* task on *BE* depends on the transmitter data rate ($\mu_{T_{LA}}$) and the amount of data to be transmitted ($w_{ACC1,AtivityCount1}$). This is multiplied by the power consumed by the transmitter ($P_{T_{LA}}$) to obtain the transmitting energy cost of a *Activity Count1*. On the other hand, processing on a sensor might be slower than processing on a *BE*. It also might consume more battery power compared to transmission of 15000 bytes which depends on power consumed in the execution of *Activity Count1* by sensor ($P_{E_{sensor}}$). Thus, we need to consider the processing cost and communication cost for each task on each resource and make task assignment decisions accordingly.

Chapter 4

Optimal Task Assignment

In this chapter, we present a static *Integer Linear Program* (ILP) based strategy, which computes the optimal task assignment for a given objective. We formulated ILP for two main objectives i.e. i) Minimizing total energy consumption of the system ii) Maximizing system lifetime. These static solutions act as a baseline of comparison for our dynamics algorithms, and also provide a static allocation for systems with a priori known characteristics.

4.1 Integer Linear Program

The goal of the first ILP (*ILP-Green*), shown in Table 4.1, is to compute the most efficient task allocation that minimizes the total energy consumption of the system. The format of the task graph is the same as the *DAG* described in Chapter 3 except that we add an explicit communication task in between two dependent sensing/processing tasks to simplify the ILP formulation. Fig. 4.1 shows a sample graph transformation. This transformation is done to represent communication task as a node. From the graph, we can determine each tasks predecessors, successors, amount of data needed from previous tasks, and the amount of data it produces.

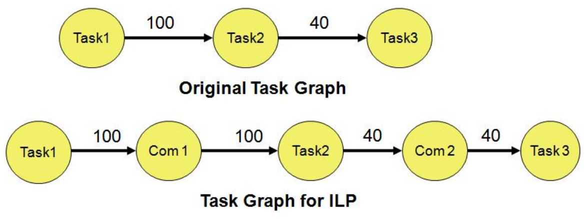

Figure 4.1: Task graph transformation for ILP

ILPGreen assigns tasks with the aim to reduce total energy without considering the battery capacity of the system components and rate of energy consumption. This might result in more work being assigned to a resource with critically low battery charge compared to other resources. As a result of this assignment that resource would discharge earlier than the other system resources and thus resulting into shorter system lifetime.

As it would be desirable to have the entire system functioning for as long as possible. We therefore explore a different ILP, called *ILPLife* that maximizes the system lifetime using a min-max formulation on the rate of energy drain relative to the available battery capacity of each node. *ILPLife* has the same constraints, constants and variables but computes the optimal assignment with the objective of maximizing system lifetime as shown in Table 4.1. Our approach to maximize the battery lifetime of the whole system is to minimize the maximum energy consumption of all the resources in the system. We want to balance the energy consumption in all the resources such that each resource consumes approximately the same percentage of energy. If one resource consumes a higher percentage of energy than the others, it would deplete its energy source first.

The complete formulation of *ILPGreen* and *ILPLife* with goals and constraints are given in Table 4.1. Among the resources *r*, resource *m* denotes the backend server and resource *m-1* denotes the local aggregator. We solve the above ILPs' for variable b_{ir}, the mapping of tasks *t* to resources *r*. Variable E_r represents the total energy consumed by a resource after tasks have been assigned to it. For *ILPGreen* our goal is to minimize sum of energy consumed by all the resources except BE i.e. sum of all E_r.

Table 4.1: ILP Objective and Constraints

***ILP*Green:** = $min(\sum_{r=1}^{m-1} E_r)$	Minimize system energy
***ILP*Life:** = $min_i\{max_r \frac{E_{r,i}}{Bat_r}\} where, r = 1, 2...m-1$	Maximize system life
(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) $E_r \leq Bat_{E_r}$	Battery Capacity Constraint
(c) $(b_{jm} + b_{jm-1}) = 1 \quad if(b_{im} = 1 \wedge pred_{ij} = 1)$	Resource Allocation Constraint 1: Predecessor tasks of a task mapped to the backend are mapped to either the backend or the local aggregator
(d) $(b_{jm-1}) + \sum_{k=1}^{m-2} b_{jk}) = 1 \quad if(b_{im-1} = 1 \wedge pred_{ij} = 1)$	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 $k = 1, 2..m-2$ $if(b_{ik} = 1 \wedge pred_{ij} = 1)$	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) $\tau_i = s_i + \sum_{r=1}^{m}\{b_{ir} * (t_{ir} + \sum_{j=1}^{n} pred_{ij} * (1 - b_{jr}) + suc_{ij} * (1 - b_{jr}))\}$	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

Our goal for *ILPLife* is to minimize the maximum percentage of energy consumption among resources *r*. In Table 4.1 E_r/E_{Br} represents the percentage of energy consumed by a resource *r* relative to the overall battery charge. All these variables and constants are explained in Table 3.1 of Chapter 3.

The constraints of the ILP are described in Table 4.1. Constraint (a) guarantees that tasks are assigned to only one resource. Constraint (b) ensures that a resource would not be assigned to more tasks than the resource has energy for. Resource allocation constraints ensure that predecessor tasks of a task are allocated to neighbouring resources that are able to communicate with each other. This ensures that a task mapped to the backend server does not have a direct predecessor that is assigned to a sensor as sensors are unable to directly transfer data to the backend server. Similarly predecessor of a task assigned to LA should be assigned to LA or respective sensor.

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 for that task. Deadline constraint ensures that tasks must finish by a given deadline.

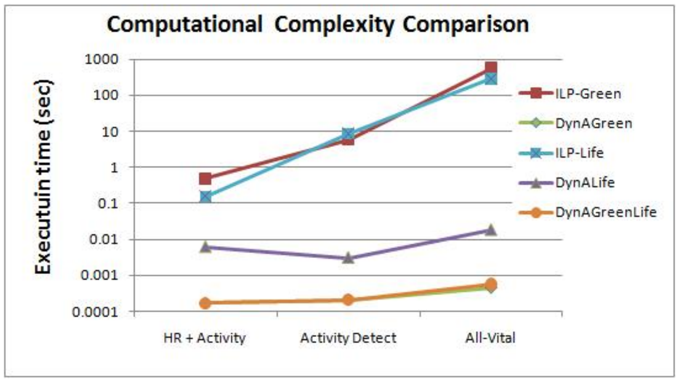

Figure 4.2: Computational overhead of ILP and dynamic algorithms

We used an open source ILP solver called *lp-solve*[11] to get the optimal task assignment for each of our tasksets. Fig. 4.2 shows the execution time of *lp-solve* on a pentium PC for various tasksets and compares it to runtime of dynamic algorithms described in next chapters. As the number of tasks in a taskset increases, execution time of *lp-solve* increases exponentially.

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, changes in task execution time due to more processing required in certain situation, and addition of some new tasks in a taskset due event based monitoring. Changes in wireless channel conditions results in changes in communication energy cost while changes in execution time of the affects computation energy cost. *LA* is a central component in our system and can easily keep track of varying system parameters without too much communication overhead. Because of this, task assignment algorithm should be run on *LA* at a regular interval or when ever the system parameters changes. Computational cost of running ILP based solution on *LA* becomes prohibitively high (e.g. 100+ seconds on a pentium PC for taskset with more than 20 tasks). Because of this ILP based solution are not practical and cannot be deployed on LA to handle dynamically changing system parameters. To address this challenge, we have designed three fast task assignment algorithms. These algorithms have very low computational overhead compared to ILP (see Fig.4.2) and hence can be run frequently to find energy efficient task assigning in changing environment. Following chapters describes these algorithms in detail.

Chapter 5

DynAGreen Algorithm

We have designed three fast and energy-efficient task assignment algorithms called *DynAGreen* (Dynamic task Assignment for Greener solution) for minimizing total energy consumption, *DynALife* to maximize system lifetime and *DynAGreenLife* to balance system energy and lifetime. These dynamic algorithms are run periodically on *LA* and utilize current energy costs of all tasks to find energy-efficient task assignment. The algorithms monitor communication costs of all the nodes and their remaining battery charge. Based on these parameters they compute the best task assignment to meet the given objective.

Energy minimization algorithm, *DynAGreen*, is described in this chapter followed by *DynALife* algorithm for system lifetime maximization in Chapter 6. *DynAGreenLife* algorithm which balances both the objectives is described in Chapter 7.

5.1 DynAGreen Algorithm

DynAGreen algorithm addresses previously defined system objective of minimizing system energy. For this algorithm we have adopted Stones method to find an assignment of tasks to wireless health system components that minimizes the total energy consumption of the distributed wireless system [16]. We use flow graph partitioning

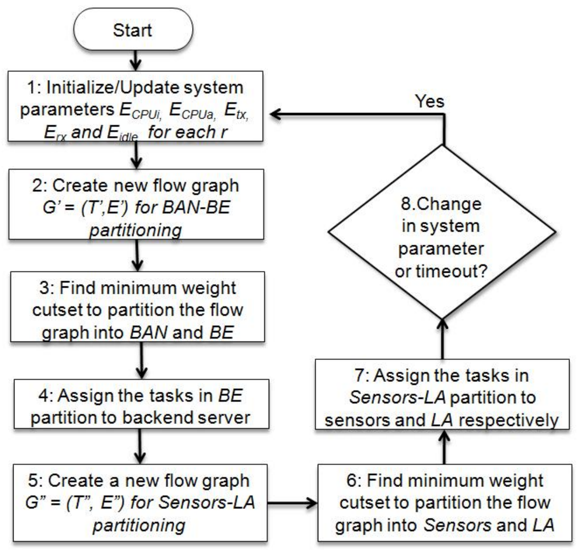
Start
1: Initialize/Update system parameters E_CPUi, E_CPUa, E_tx, E_rx and E_idle for each r
2: Create new flow graph G' = (T',E') for BAN-BE partitioning
3: Find minimum weight cutset to partition the flow graph into BAN and BE
4: Assign the tasks in BE partition to backend server
5: Create a new flow 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 5.1: DynAGreen Algorithm

technique to partition tasks between different components (server, LA, sensors) in a hierarchical fashion. The hierarchical partitioning is applied in two steps. In the first step, a flow graph constructed from given task graph is partitioned between an infinite energy *BE*, and a super-node referred as *BAN* that comprises 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* and partitioning is done using a maxflow/mincut algorithm. The minimum weight cutset represents the energy cost if tasks are partitioned by the cutset and this provides partitioning of tasks to minimize energy between the server and the *BAN*. Subsequently, in a second step, we partition the tasks within the *BAN* between the *LA* and the sensors to minimize the utilization of energy within the *BAN*. Following are the details of each step in the algorithm.

1. Computation 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 energy consumed by a CPU in active and idle states respectively and E_{tx}, E_{idle}, and E_{rx} represent energy consumed by radio in transmit, receive and idle state respectively. In the first run of the

algorithm they are initialized with default values and in subsequent runs they are updated based on the radio interface monitoring.

2. Tasks are partitioned between *BAN* and *BE* such that 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 given task graph into a flow graph with communication and computation energy cost as flow values, *BAN* as a source node and *BE* as sink node such that minimum weight cutset represents the total of computation and communication energy cost for the partition. Formal description of this transformation is explained in Section 5.1.2.

3. We find the minimum weight cutset to partition the constructed flow graph into *BAN* and *BE* partition. In [16] Harold Stone proves that such a minimum weight cutset provides optimal task assignment for a two processor system. Therefore, we obtain *BAN-BE* task partition that results in minimum energy cost. The partitioning process is described in detail in Section 5.1.1.

4. 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 on *Sensors* and *LA* in 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}.

5. All the sensors are collectively represented by a single source node,*Sensors*. *LA* is represented by a single sink node *LA* in *Sensors-LA* flow graph. As per our system model, all tasks are traced back to one or more sensing tasks by following their predecessor chain. These sensing tasks are pre-assigned to sensors or *LA*. A processing task that receives its input from more then one sensors can not be assigned to a sensor since as per our system model sensors can not communicate with each other directly. Such a task has to be assigned on LA or BE. By

assigning appropriate weights on the edges in flow graph, we ensure that such tasks cannot be in *Sensors* partition. This also implies that if a task is in Sensors partition, it should receive its input from only one sensor and we can find that sensor by tracking the task's predecessors. Similar to *BAN-BE* partitioning, we partition tasks between *Sensors* and *LA* such that total of computation and communication energy cost of the system is minimum. To achieve this objective we create *Sensors-LA* flow graph from remaining tasks with communication and computation energy cost as flow values such that minimum weight cutset represents the total of computation and communication energy cost for the partition. Section 5.1.2 formal description of this transformation

6. We find the minimum weight cutset to partition the constructed flow graph into *Sensors* and *LA* partition. This *Sensors-LA* task partition results in a task assignment that has minimum energy cost. Sensor-LA partitioning is explained in detail in Section 5.1.2.

7. Tasks in *Sensor* partition are assigned to respective sensors and tasks in *LA* partition are assigned to *LA*. Even though *Sensors* node in flow graph represents multiple sensors, after finding mincut, we can assign tasks in *Sensors* partition to appropriate sensor.

8. To detect the changes in system characteristics, communication cost 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 LCM (Least Common Multiple) of the period of all tasks in the taskgraph. Go back to step 1 when ever significant change(30%) in communication cost parameters is detected.

We used implementation of mincut/maxflow algorithm proposed in [5]. Authors have shown that their algorithm beats all existing polynomial time mincut algorithms. In Fig. 4.2, we compare the execution time of *ILP-Green* and *DynAGreen* when run

Table 5.1: Weights of edges in BAN-BE partition

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

on a PC. The computational time of *ILP-Green* is significantly 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 Fig. 4.2 . Because of this low computational overhead compared to *ILP-Green*, our approach enables frequent execution to address dynamically changing system parameters.

5.1.1 BAN-BE partition

In this section the *BAN-BE* partitioning of the *DynAGreen* algorithm is explained in detail with an example. Following steps explains the partition process.

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

- E_{BAN} is a set of edges from *BAN* to the each node *t* in *T*. Weight of edge (BAN, t) is equal to computational cost of task *t* on *BE* and it is defined by $w(BAN, t)$ in Table. 5.1. 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 computation cost of performing this task on *BE* is added to weight of the cutset. We consider *BE* to have unlimited energy and hence computation energy cost of performing a task on *BE* is set to 0 in Table. 5.1. If a task *t* is bound to sensor or *LA* node, we need to ensure that a minimum weight cutset does not include edge (BAN, t). We achieve that objective by setting the weight of the edge (BAN, t) to ∞ as per Table. 5.1.

- E_{BE} is a set of edges from each node *t* in *T* to *BE* with weight of edge (t, BE) equal to computational cost of task *t* on *LA* as defined by $w(t, BE)$ in Table. 5.1, where $E_{CPU}(t, LA)$ is the computation energy cost of running task *t* on *LA*. This weight assignment has similar rationale to (BAN, t) weight assignment. If a mincut includes edge (t, BE) then it means that *t* is in *BAN* partition and computation cost of performing this task on *BAN* is added to weight of the cutset. Note that in Table. 5.1 we use *LA*'s parameters to calculate computation energy cost as *BAN* is represented by *LA* during this stage of the algorithm. We enforce this edge to be part of minimum weight cutset by setting its weight to 0 as per Table. 5.1 if the task is bound to a sensor or *LA*. Similarly we enforce that this edge is not part of minimum weight cutset by setting its weight to ∞ if the task is bound to *BE*.

- Edge (t_p, t_s) in *E* represent the inter-task communication link between predecessor task t_p and its successor task t_s. Weight of edge (t_p, t_s) is equal to communication cost of sending t_ps output from *LA* to *BE* and is defined by $w_{wan}(t_p, t_s)$ in Table. 5.1, where $E_{radio}(t_p, LA)$ is communication energy consumed by task t_p on resource *LA*. Note that we use *LA*'s system parameters to compute communication energy cost of *BAN*.

- We find the minimum weight cutset to partition the constructed flow graph into *BAN* and *BE* partition. Once we find the minimum energy cost partition between *BAN* and *BE*, tasks in *BE* partition are assigned to *BE*.

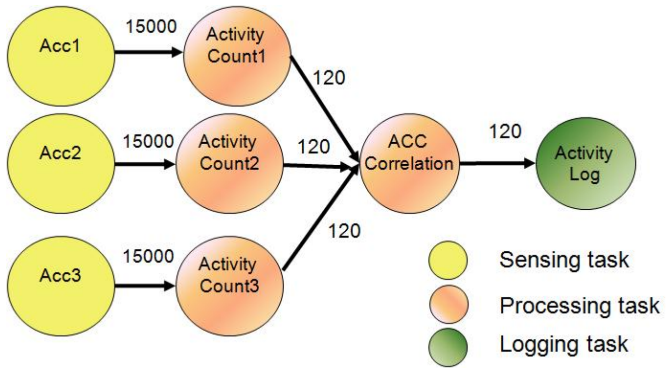

Figure 5.2: Example Taskset

Example: We use the taskset shown in Fig. 5.2 to provide an intuitive explanation of this part of the algorithm. Fig. 5.3 shows flowgraph G' for *BAN-BE* task partitioning constructed from graph G Fig. 5.2. Notice that we added two new nodes *BAN* and *BE* to the original graph. All the edges from *BAN* to tasks have computation cost of running tasks on *BE* as their weights and that cost is defined in Table. 5.1. We set computational cost of running a task on *BE* to 0 since it does not affect battery lifetime of sensor or *LA*. Hence as per the Table. 5.1 if given task is not bound to sensor or *LA*, we set the weight to 0. For example edges from *BAN* to *tasks 3, 4, 5, 6 and 7* have weight of 0 in Fig. 5.3. In case if a task is bound to specific sensor or *LA*, we want to make sure that edge between *BAN* and task doesnt become part of minimum cutset. This is ensured by assigning weight of ∞ to such an edge. 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 computation costs of running them on *LA* and their weights and are set as per Table. 5.1. For example, edge from task 3 to *BE* has energy cost of $E_{CPU}(3, LA)$. For tasks that are bound to either sensors or *LA*, we want to ensure that edge from that task to *BE* is part of minimum cutset and hence we set it 0 in Table. 5.1. If a task is bound to *BE* we want to 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.

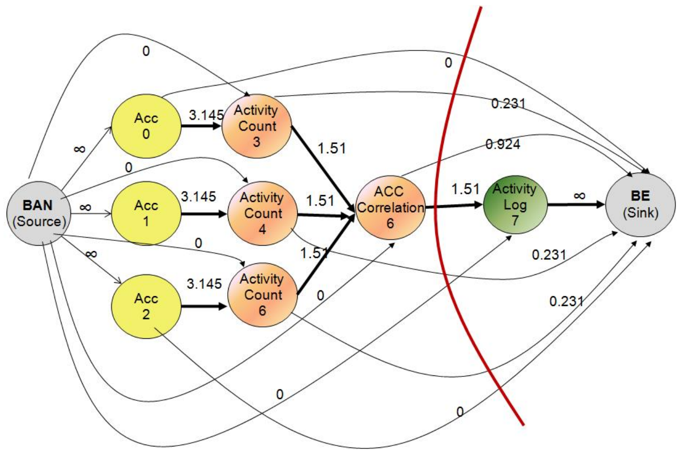

Figure 5.3: BAN-BE partition

Weights of edges between the tasks represent communication cost. For example weight of the edge between *task 0* and *task 3* represent amount of energy required to send output of *task 0* from *LA* to *BE*. We use Table. 5.1 for this. Note that we use *LAs* wide area network radio energy parameters for this cost calculations since we are trying to find trade off 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). Fig. 5.3 shows minimum weight cut set for the example task set and this provides energy cost optimal assignment for *BAN-BE* partition as per [16]. As a result of this, *task 7* is assigned to *BE*.

5.1.2 Sensor-LA partition

Partitioning of tasks between sensors and *LA* is explained below:

- We create a new flow graph G'' for *Sensors-LA* partitioning using all tasks in *BAN* partition and then adding $Sensors, LA$ and a set of proxy tasks. For all the tasks in T_{BAN} whose successors are assigned on *BE*, we create new task nodes which are bound to *LA* and act as transmission tasks to *BE* and are represented by set

Table 5.2: Weights of edges in Sensor-LA partition

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

T_{proxy}. These proxy tasks are added to represent the communication energy cost of forwarding output of a predecessor task from a Sensor to *LA* so that *LA* can forward it further to *BE*. Edges of this new graph consist of following edge sets: E_{BAN} representing communication energy cost, $E_{Sensors}$ representing computation cost of *LA*, E_{LA} representing computation energy cost of sensors and E_{proxy} to enforce proxy task assignment on *LA*. So formally this new flow graph is defined as $G'' = (T'', E'')$, where $T'' = (T_{BAN} \cup T_{proxy} \cup \{Sensors, LA\})$ and $E'' = (E_{BAN} \cup E_{Sensors} \cup E_{LA} \cup E_{proxy})$.

- $E_{Sensors}$ is a set of edges from Sensors node to the each node *t* in $T_{BAN} \cup T_{proxy}$ with its weight equals to computational cost of running task *t* on *LA* and it is defined by $w(Sensors, t)$ in Table 5.2. The rationale behind this weight assignment is if a mincut includes edge $(Sensors, t)$ then it means that *t* is in *LA* partition and 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 need to ensure that a minimum weight cutset does not include edge $(Sensors, t)$. We achieve that objective by setting the weight of the edge $(Sensors, t)$ to ∞ as per Table 5.2. Similarly if a task *t* is bound to *LA*, we guarantee inclusion of this edge in minimum weight cutset by setting $w(Sensors, t)$ to 0.

- E_{LA} is a set of edges from each node *t* in $T_{BAN} \cup T_proxy$ to *LA* with its weight equals to computational cost of running task *t* on *Sensors* and is defined by $w(t, LA)$ in Table 5.2, where $E_{CPU}(t, Sensor)$ is the computation energy cost of running task *t* on a sensor. As noted earlier, if a task receives its input from multiple sensors, it can only be assigned on *LA* and hence is considered bounded on *LA*. Any task *t* bounded to *LA* has $w(t, LA)$ set to ∞ to ensure that this edge does not become part of minimum weight cutset. If a task *t* is bound to sensor, we set the $w(t, LA)$ to 0 and that ensures edge (t, LA) to be part of minimum weight cutset. For a task *t* that is not bound to a sensor or *LA*, we find the sensor generating its input (by traversing through task's predecessor change finding root task and its associated sensor) and we use that sensor's CPU parameters for setting computation energy cost of the edge.

- E_{proxy} is a set of edges (t_p, t_s) where $t_p \in T_{BAN}$ and $t_s \in T_{proxy}$ and E_{BAN} is a set of edges (t_p, t_s) where $t_p, t_s \in T_{BAN}$. Weight for edge $(t_p, t_s) \in E_{BAN} \cup E_{proxy}$ is the communication cost of transmitting t_ps output from its source *Sensors* to *LA* and is given by $w_{ban}(t_p, t_s)$ in Table 5.2 where $E_{radio}(t_p, Sensor)$ is communication energy consumed by task t_p on resource *Sensor*.

- We find the minimum weight cutset to partition the constructed flow graph into *Sensors* and *LA* partition. This obtained *Sensors-LA* task partition results in minimum energy cost .

- Tasks in *Sensor* partition are assigned to respective sensors and tasks in *LA* partition are assigned on *LA*.

Example: To explain this partitioning procedure better we use the same example used in 5.1.1. Fig. 5.4 shows flowgraph G'' for Sensors-LA task partitioning constructed from graph G' i.e. Fig. 5.3 as per above mentioned steps. *Sensor* and *LA* nodes are added to tasks of *BAN* partition. *Sensor* and *LA* nodes are added to tasks of BAN partition. In our example graph, we add *ACC Correlation Proxy task 6* because its successor is assigned on *BE*. The idea behind adding this task is to factor in costs to transmitting

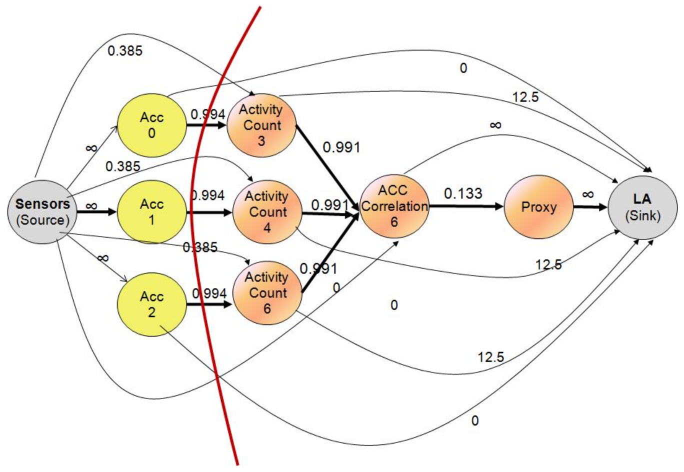

Figure 5.4: Sensor-LA partition

task 6's output to *LA* in case *task 6* gets assigned to a sensor. We bind all the proxy tasks to *LA* since they are a place holder with no computation cost. Since in our system model we do not assume inter-sensor communication link, we cannot assign a task to a sensor if the task receives its input from multiple sensors sources and hence we bind it to *LA*. In our example, *ACC Correlation* is such a task and hence gets bound to *LA*. We set the weights as per Table 5.2. Fig.5.4 shows minimum weight cut set for *Sensor-LA* partition. All tasks on in *LA* partitions are assigned to *LA*. Note that if a task and tasks proxy both are in *LA* partition, we ignore the proxy task since it is only required if its corresponding task is not assigned on *LA*. For Sensors partition, we find a sensor node for a task by traversing it 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* will be assigned to a sensor to which *task 0* is bound to if *task 3* would have been in sensor partition.

Chapter 6

DynALife Algorithm

This chapter provides an overview of the dynamic task assignment algorithm *DynALife* designed to maximize system lifetime. ***DynALife*** is a fast and energy-efficient heuristic task assignment algorithm. System lifetime is defined as the time from the start of the system toll the time first battery operated device dies. *DynALife* was introduced by the thesis author and other co-authors as the algorithm *DynAHeal* in [13]. However, for clarity of presentation, we shall refer to DynAHeal as DynALife in the rest of this paper. Cellphones and sensors can have different battery charge capacity and different rate of energy consumption. For example, cellphones typically have larger batteries then small sensors and they consume much more energy compared to sensors due to LCD, speaker, and other peripherals. To maximize the system lifetime, we need to assign tasks such that node with least battery charge gets least amount of work. *DynAGreen* algorithm does not consider remaining battery charge of a device while assigning tasks and hence minimizing energy consumption using *DynAGreen* algorithm does not always maximize system lifetime. For example, lets say at the time of assignment shown in Fig. 5.4 computed by *DynAGreen*, *LA* has only 10% battery charge remaining while an all sensors have 80% of their battery charge remaining. In this scenario we can reduce the workload on *LA* by assigning *Activity Count* tasks on sensors instead of *LA* and this way increase the system lifetime by increasing *LA*'s battery lifetime. *DynALife* algorithm addresses such a scenario.

Fig. 6.1 outlines the heuristic algorithm. We assign the tasks one at a time starting from source tasks. For each assignment we select a task assignment such that it maximizes the system battery lifetime compared to all other task assignments possible at a given stage and it also meets task deadline. We perform this operation iteratively until all tasks are assigned. After the initial assignment we iterate over the assignments and re-examine each tasks assignment to see if an alternative assignment can improve system lifetime. We end the algorithm when further iteration does not improve system battery lifetime. We next provide a detailed description of *DynALife* algorithm. Following are definitions of set 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
- $E \subseteq R$: set of eligible tasks. All remaining tasks whose predecessors have already been assigned

Algorithm steps:

1. Assign the tasks of set *P* to specified nodes of the system. *P* is used to put user-defined constraints on task assignment and typically contains sensing tasks bound to specific sensors and data logging tasks bound to the *BE*.
2. Update the sets *A, E* and *R* as per their definition.
3. Compute *Battery Life Estimation* table for all tasks in *E*. See example of such a table in Table 6.1. We estimate a sensors battery lifetime and the *LA*s battery lifetime for each possible assignment of task (on sensor, *LA* or *BE*). We then

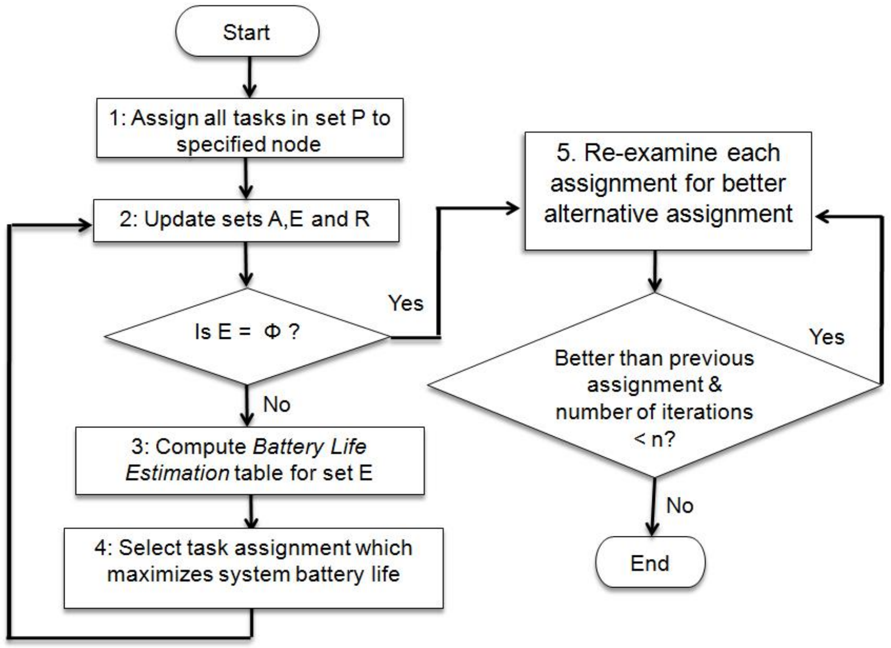
Start
1: Assign all tasks in set P to specified node
2: Update sets A,E and R
Is E = Φ ?
Yes
No
3: Compute Battery Life Estimation table for set E
4: Select task assignment which maximizes system battery life
5. Re-examine each assignment for better alternative assignment
Better than previous assignment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

Figure 6.1: DynALife Algorithm

compute the system battery lifetime as minimum of *sensor's* battery lifetime and *LA's* battery lifetime for each possible assignment. The maximum of these three system lifetime values is the maximum system battery lifetime that we can achieve for the given task. We store the node ID of the best task assignment as *best assignment node ID*. We also compute *deadline slack* for each possible task assignment. If a particular task assignment does not meet the deadline this value will be negative so we do not consider that assignment possibility.

4. Sort the *Battery Life Estimation* table based on maximum system battery lifetime and select the task with the longest system battery lifetime for assignment. This results in the maximum system lifetime achievable at this stage. After this assignment, we go to step 2 and the process is repeated until all the tasks are assigned.

5. Iterate over each task assignment to find a better solution by reexamining assignment of an individual task starting from the source. While considering an

Table 6.1: Battery life 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**	**Best assignment node ID**
1	(600, 1200, 1020)	(800, 1500, 1500)	(600, 1200, 1020)	(500, 350, 200)	(1200)	LA

assignment for a task during initial assignment, we do not know where its successor tasks will be assigned. This might result in missing a better assignment choice which can be found in this step. We deassign the task being reexamined and compute the *Battery life estimation* table for it. We use the b*est assignment node ID* as the new assignment for the task. We continue reexamining all the task assignments until we reach the sink tasks. If the new assignment is better than the previous assignment, then we keep that and repeat *step 5*; otherwise we terminate the algorithm. We bound the number of iterations to *n* to limit execution time.

We compute Table 6.1 each time before we assign a task. For a taskset consisting of n tasks, the table computation is performed n times. Number of rows in the table could be n in worst case scenario. Each row of the table is computed in constant time. Computed table is sorted resulting in overall time complexity of $O(n^3log(n))$ for algorithm loop before step 5. For step 5, the worst case time complexity of repetitions is $O(n^2)$. In practice we have not observed more then 2-3 repetitions of step 5. Hence complexity of the entire *DynALife* algorithm is $O(n^3log(n))$. With this low overhead of the algorithm compared to the *ILPLife* (which is NP-hard), the *DynALife* algorithm can be run dynamically to address dynamically changing system parameters.

DynALife targets the objective specified by *ILPLife* to extend the system lifetime in the system to be as long as possible, while executing the required tasks in the system.

However, an algorithm such as *DynALife* that focuses purely on system lifetime has a drawback. It targets the most critical energy resource node (either the *LA* or one of the sensors) in a system with *N* 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. In this process it completely ignores how energy is distributed between the remaining *(N-1)* nodes, and it can inefficiently assign tasks to these remaining *(N-1)* nodes resulting in a faster depletion of energy in this nodes. Since all nodes are energy constrained, it is necessary to minimize the energy utilization in these remaining *(N-1)* nodes while extending the system lifetime as much as possible. Thus, we need a dynamic task assignment algorithm that would maximize 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.

Chapter 7

DynAGreenLife Algorithm

In this chapter we describe *DynAGreenLife* that jointly optimizes for both system energy and system life. *DynAGreenLife* is a variation of *DynAGreen* algorithm and mainly differs in *Sensors-LA* partition. In *BAN-BE* partition, we always prioritize BAN and hence differential battery charges of sensors and LA do not make any difference. For *Sensors-LA* partitioning, *DynAGreenLife* takes into account the remaining battery charges of sensors and LA in addition to computational and communication energies to determine weights of the flow graph. If a sensor's battery charge is relatively low compared to *LA*, we increase the weight of edges from that sensors to various related tasks compared to edges from those tasks to *LA*. This increases the chances of sensor to task edges being part of minimum weight cutset and hence assigning more work to *LA* because of it has more battery charge even though assigning given task to sensor would result in optimal system energy. In this manner, during partitioning, the criticality of a resource that has lower energy compared to sensors or *LA* is highlighted by a higher weight on related edges in flow graph. By increasing the weight based on the relative lower energy availability of a resource, the likelihood of task assignment to the resource is reduced and trade of between system energy and system lifetime is achieved. This change in the algorithm results in a balance between system energy and system lifetime optimization.

DynAGreenLife has the same steps of *DynAGreen* (shown in Chapter 5) except step 5. In first step we initialize or update the energy cost parameters. In second and third step, we create *BAN-BE* flowgraph and find the mincut. In 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 weight of an edge by multiplying its *DynAGreen* weight with a factor which depends on the relative battery charge of the sensor in question and *LA*. Equations (7.1-7.3) define weights of edges in flow graph utilizing definitions given in Table. 5.2 of Chapter.5. $E_{bat}(r)$ is the remaining battery charge of resource *r*.

Equation 7.1 defines weight of 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 7.2 defines weight of edge from a task *t* to *LA* in flow graph. To understand the effect of battery charge based multiplying factors, consider a scenario in which sensors battery is low compared to *LA*'s battery. In this case, $w(Sensors, t)$ shown in Table. 5.2 of Chapter 5, is multiplied with a lower multiplying factor compared to $w(t, LA)$, resulting in relatively lower weight to edge from *Sensors* to task *t*. Similarly $w_{ban}(t_p, t_s)$ is multiplied by a higher multiplying factor increasing its weight. This results in increasing chances of task *t* being part of *LA partition*. Equation 7.3 defines weight of edge (t_p, t_s) from a predecessor task t_p to successor task t_s. Note that since we increase weight of communicating edges as well it will only favour assignment of a task to *LA* if the cost of sending tasks input from sensor to *LA* does not result into higher sensor energy drain.

$$w^{'}(Sensors, t) = w(Sensors, t) * \frac{E_{bat}(s) + E_{bat}(LA)}{E_{bat}(LA)} \tag{7.1}$$

$$w^{'}(t, LA) = w(t, LA) * \frac{E_{bat}(s) + E_{bat}(LA)}{E_{bat}(s)} \tag{7.2}$$

$$w^{'}_{ban}(t_p, t_s) = w_{ban}(t_p, t_s) * \frac{E_{bat}(s) + E_{bat}(LA)}{E_{bat}(LA)} \tag{7.3}$$

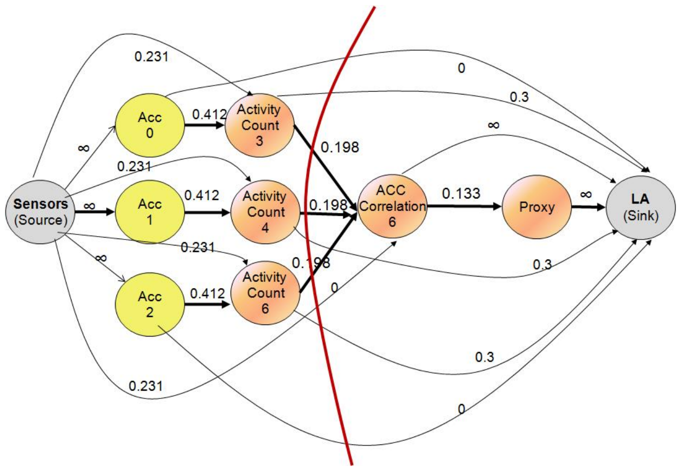

Figure 7.1: Sensor-LA partition

Example:

We use the same example of taskset shown in Fig. 5.2 as we did in *DynAGreen* algorithm. The algorithm first performs *BAN-BE* partitioning and then *Sensors-LA* partitioning. The *BAN-BE* partition is same as *DynAGreen* and shown in Fig. 5.3. The difference between *DynAGreenLife* and *DynAGreen* algorithm is in *Sensors-LA* partition because *DynAGreenLife* considers current battery energy of the resources unlike *DynAGreen*. *DynAGreenLife* computes the weights as shown in equations 7.1, 7.2 and 7.3. In our example *LA* has critical battery and it multiplies a factor $\frac{E_{bat}(s)+E_{bat}(LA)}{E_{bat}(LA)}$ to weights of edges from *Sensors* to task nodes and edges between the tasks. While the weights from tasks to *LA* is multiplied by $\frac{E_{bat}(s)+E_{bat}(LA)}{E_{bat}(s)}$. As *LA* has critical battery $\frac{E_{bat}(s)+E_{bat}(LA)}{E_{bat}(LA)} > \frac{E_{bat}(s)+E_{bat}(LA)}{E_{bat}(s)}$. Thus, weights among the edges and edges from *Sensors* node to tasks would be higher such that a mincut does not pass through these edges and more tasks are assigned on sensors. In Fig. 7.1 the weights of edges from *task 0* to *task 3*, *task 1* to *task 4* and *task 2* to *task 5* is higher than weight on edges from *tasks 3,4,5* to *LA* node. This additional factor helps to assign less work on the 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* unlike *Sensors-LA* partition in *DynAGreen*.

Similar to *DynAGreen*, we compare execution time of *DynAGreenLife* with our static *ILP* based solutions in Fig. 4.2. *DynAGreenLife* is computationally efficient and hence is more suitable for dynamic assignment/reassignment of tasks. In the next chapter we demonstrate the ability of our algorithms to adapt at runtime to changing system characteristics.

Chapter 8

Results

In this section, we evaluate our dynamic task assignment techniques in terms of its effect on both the system battery lifetime and energy consumption. We demonstrate results for three different wireless health care system task sets and analyze the performance of our dynamic algorithms. We compare *DynAGreen*, *DynALife* and *DynAGreenLife* algorithm with static solution given by *ILPGreen* and *ILPLife* and All-On-BE strategy under static conditions. We also demonstrate the ability of our algorithms to adapt at runtime to changing system conditions. Later we show how *DynAGreenLife* algorithm balances both system lifetime and system energy by comparing it with *DynAGreen* and *DynALife* algorithms.

8.1 Experimental Setup

We used three main task graphs for our experiments which are described in Table. 8.1. Tasks in Table. 8.1 represent applications used in preventive healthcare systems such as PALMS [7]. Each task has an arrival rate, number of instructions to be executed and number of output bytes. All the details for each task graph are given in Tables. 8.2, 8.3 and 8.4. Input column specifies how many bytes are consumed by a task on each occurrence. Output column specifies the number of bytes produced by the task. Number of instructions are estimated based on the amount of processing required for each sample. For example, ECG task produces 500 bytes each second and if 60 instructions are required to produce 1 byte, total number of instruction for each occurrence of ECG task is 30000. Outputs of the sensing tasks were determined by the commercially available sensor's output.

Table 8.1: Experimental Workload

Taskgraph	**# of sensors**	**# of tasks**	**Application**
HR + Activity	2	6	ECG sensor detects heart rate per minute. While accelerometer keeps track of activity
Activity Detect	3	8	Detects a persons activity using three accelerometers.
All-Vital	5	20	Log all vital signs like heart rate, blood pressure, activity in addition to location

HR+Activity task graph shown in Fig.8.1 keeps track of hear rate and activity of the user during the day. This task graph has the ECG and Accelerometer sensors as two sources. The hear rate and activity data is finally combined by *ActivityHR* Log task and stored on backend device every minute. *Activity Detect* task graph uses three accelerometers as shown in Fig.8.2 to predict the activity of the person wearing them. These accelerometers are worn on arm, waist and thigh to determine whether the person is steady, walking or running. Such task graphs are mainly used in preventive healthcare and weight management applications. *Activity Count1,2,3* filters and processes data from individual source accelerometers while *Activity Correlation* combines the output of each *Activity Count* task and runs some algorithms to determine the posture of the user. Task graph *All-Vital* senses, processes and logs two important vital signs of a patient i.e. hear rate and blood pressure along with current GPS location and activity. All these values are logged into a remote server for healthcare professionals to study later.

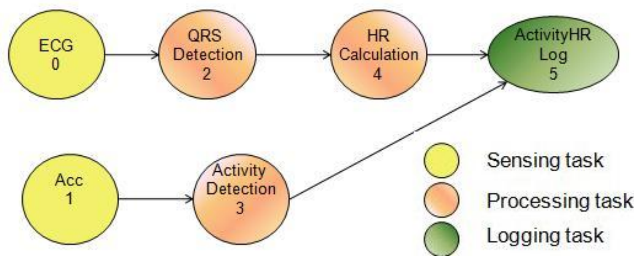

Figure 8.1: Task graph: HR + Activity

Table 8.2: Taskgraph: HR+Activity

Task	**input (bytes)**	**output (bytes)**	**# of instructions**	arrival rate
ECG	0	500	30000	1 sec
ACC1	0	600	15000	1 sec
QRS detection	1500	600	3750000	3 sec
Activity Detection	1800	36	55000	3 sec
HR calculation	600	50	12000000	3 sec
ActivityHR log	1720	0	3000	1 min

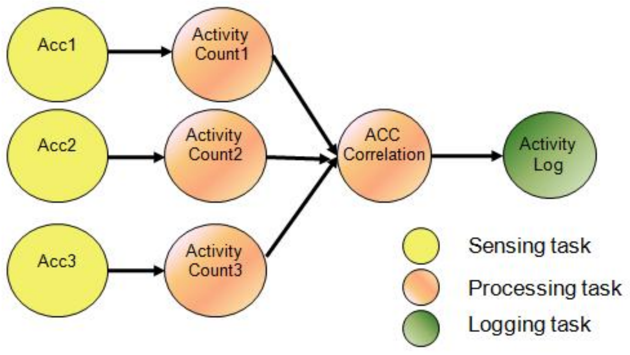

Figure 8.2: Task graph: Activity Detect

We use Qualnet [15] a state of the art discrete event wireless network simulator. Qualnet provides accurate wireless channel models, a variety of wireless protocols along with their energy, battery and mobility models. We added a simple model for computational energy consumption and used CPU current loads provided in MicaZ and Intel XScale processor specifications into Qualnet. We modeled sensors nodes as MicaZ nodes with Zigbee(802.15.4) radio in Qualnet. The local aggregator is modeled as a

Table 8.3: Taskgraph: Activity Detect

Task	**input (bytes)**	**output (bytes)**	**# of instructions**	arrival rate
ACC1	0	300	20000	500 msec
ACC2	0	300	20000	500 msec
ACC3	0	300	20000	500 msec
Activity count1	600	60	500000	1 sec
Activity count2	600	60	500000	1 sec
Activity count3	600	60	500000	1 sec
ACC correlation	180	80	80000	1 sec
Activity log	80	0	0	1 sec

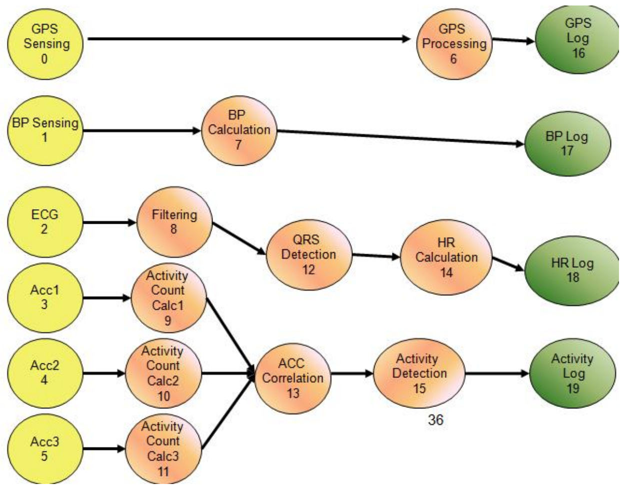

Figure 8.3: Task graph: All-Vital

UMTS-UE (User Equipment i.e. Handset) with an additional Zigbee radio interface and CPU speed of 400MHz. Table 8.5 provides key parameters configured in Qualnet. We use an energy model for MicaZ node provided by Qualnet and specify a constant transmit power of 3dBm. For UMTS, we configured a generic energy model and specified -10dBm as the minimum transmit power and 30dBm as the 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. Other parameters for the simulator are given in

Table.8.5 .

The handset is connected to the Backend Server via the UMTS network and with sensors via Zigbee radio. Tasks assigned to a resource send data to the next resource over UDP if any of its successor tasks is not assigned on the same resource. We implemented logic for periodic task execution, data transmissions, radio link parameter measurement and reporting, task assignment control messaging and execution of the dynamic algorithms at the application layer. We created an ILP for each task graph and used an open source ILP solver called *lp-solve*[11] to get the optimal task assignments.

Table 8.4: Taskgraph: All Vital

Task	**input (bytes)**	**output (bytes)**	**# of instructions**	arrival rate
GPS	0	60	24000	1 sec
ECG	0	150	22500	500 msec
ACC1	0	300	20000	500 msec
ACC2	0	300	20000	500 msec
ACC3	0	300	20000	500 msec
BP	0	250	18000	1 sec
GPS process	3600	108	100000	1 sec
ECG filter	300	225	700000	1 sec
Activity count1	15000	300	150000	1 min
Activity count2	15000	300	150000	1 min
Activity count3	15000	300	150000	1 min
BP calc	30000	36	40000	2 min
QRS detection	225	40	400000	1 sec
ACC correlation	900	40	300000	1 min
HR calc	2400	36	12000000	1 min
Activity detection	40	36	1500000	1 min
GPS log	108	0	0	1 min
HR log	36	0	0	1 min
Activity log	36	0	0	1 min
BP log	36	0	0	2 min

Table 8.5: Simulator parameters

Components	Characteristic	Value
Zigbee radio	Propagation Channel Frequency	905 MHz
	Propagation Pathloss Model	Two-Ray
	PHY 802.15.4-TX Power	3.0dBm
UMTS radio	Propagation Channel Frequency	2.4 GHz
	Propagation Pathloss Model	Two-Ray
	PHY UMTS MAX TX Power	30.0 dBm
	PHY UMTS MIN TX Power	-10.0 dBm
	Power Amplifier Inefficiency Factor	6.5
	Transmit Power Consumption	100.0 mwatt
	Receive Power Consumption	130.0 mwatt
	Idle Power Consumption	120.0 mwatt
	Sleep Power Consumption	0.0 mwatt
	Supply Voltage	3.0 V
Sensor CPU	Active Current	50 mAmp
	Frequency	8 MHz
	Voltage	3.0 V
LA CPU	Active Current	308.33 mAmp
	Frequency	400 MHz
	Voltage	3.0 V

In all our experiments we measure and compare system lifetime and total energy consumption. System lifetime is defined as the time from the start of the system until the time first device runs out of energy. Total energy consumed by all the battery operated devices of the system till the system dies is termed as total energy consumption.

8.2 Static Conditions

In the initial set of experiments we assume that static system parameters such as the arrival rate of tasks, computational complexity, and wireless channel conditions remain constant. In the first set of experiments we set the initial battery charge for each sensor of 100 mAh and 300 mAh for the cellphone. We measure system lifetime achieved by task assignment given by the ILPs and three dynamic algorithms and compare it against state of the art strategy which streams all the data to BE for processing (All-On-BE). In static conditions both ILPs come up with the same task assignments for our taskgraphs. In Fig 8.4 we show percentage improvement in system battery lifetime based on the task assignments as determined by the ILPs' with respect to All-On-BE. For the *All-Vital* task graph that has a higher number of processing tasks, ILPs perform 60% better than All-On-BE. On average, the ILP improves the system battery lifetime by 37%. These results shows that task assignment can significantly impact the system lifetime. We also run dynamic algorithms under same conditions and observe that the proposed dynamic algorithms perform within 0.001% of the ILP and thus perform very close to the optimal solution.

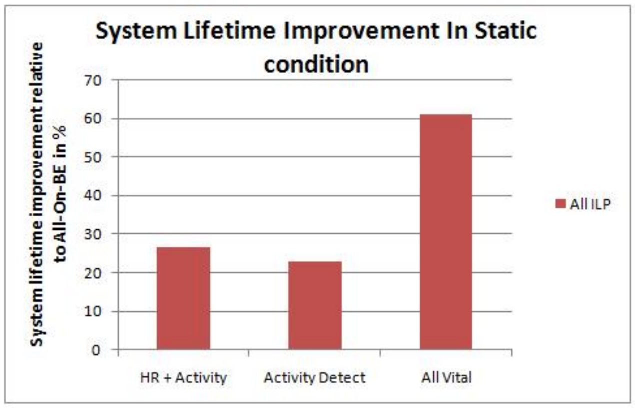

Figure 8.4: Percentage improvement in system battery lifetime achieved in static conditions compared to processing all data on BE(*All-On-BE*

We measured the average energy consumed by each of the task assignments in the above set of experiments. Fig 8.5 displays the percentage energy savings achieved by both ILPs and all three dynamic algorithms compared to the All-On-BE strategy. All dynamic algorithms gained up to 42% energy savings which is similar to that obtained by the optimal assignment given by *ILPGreen* and *ILPLife*.

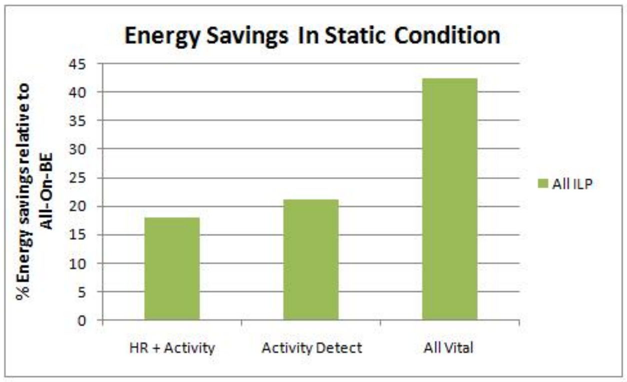

Figure 8.5: Percentage reduction in energy consumption achieved in static conditions compared to processing all data on BE(*All-On-BE*)

The ILPs and *DynAGreen*, *DynALife* and *DynAGreenLife* algorithms perform the same both lifetime wise and energy consumption wise. One key reason for this is that we assume fully charged sensors and cellphone at the start of the experiment. In general, this is not always true as sensors and cellphones have different battery consumption rates. A cellphone battery is consumed rapidly as it performs other tasks in addition to health monitoring tasks unlike sensors which often have a single function. Also cellphones are charged more often as compared to sensors.

Consider a situation where the cell phone has critically low battery while the sensors are almost fully charged. Algorithm such as *ILPGreen* or *DynAGreen* does not take into consideration the actual energy level of these components as it purely optimizes for minimizing energy consumption of the overall system. If we use these algorithms, it will merely focus on ensuring that the distribution of tasks among the various components (sensors, cellphone, back-end) is such that the total energy is minimized regardless of the remaining battery lifetime of the different devices. On the other hand, *ILPLife* or *DynALife* optimize for system lifetime by considering the actual battery levels of the devices, and try to minimize the load on the components that have very low battery level. However, such an algorithm can choose to allocate tasks in a manner that depletes energy faster in the overall system. *ILPLife* with its min-max formulation, focusses on the least critical resource in the system by allocating minimum tasks on the critical resource. Such an algorithm will tend to neglect the impact of task assignment to other nodes while trying to reduce the load on the most critical resource. Effectively, more energy is likely to be wasted if the task assignment focuses purely on increasing system-lifetime and in increasing the life-time of a most critical resource. A more balanced algorithm, such as *DynAGreenLife*, is likely to address both the criticality of the resources, and the overall energy consumed to perform the desired tasks.

We demonstrate the ability of our dynamic algorithms to adapt to different battery charge and battery consumption rates in the next set of experiments. We assume that the battery level is at 300 mAmp for each sensor and that the cellphone battery level is at 100 mAmp. Fig. 8.6 shows the battery lifetime improvement as obtained by the ILPs and the DynA family of algorithms compared to a typical All-On-BE strategy. It is evident from these experiments that task assignment techniques make a huge difference up to 140% longer battery lifetime compared to performing all processing on BE as shown in Fig. 8.6. The results also show that *ILPLife*, *DynALife*, and *DynAGreenLife* achieve higher system lifetime than *ILPGreen* and *DynAGreen*. This is because the task partitioning in *ILPGreen* and *DynAGreen* is purely focused on a green partitioning of the tasks without taking any consideration of the available battery charge of the system devices. It can also be observed that *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 to better optimize for system-lifetime.

Although the goal is to attempt a graceful reduction in energy proportionately across devices to increase system lifetime, task allocation to devices is discrete in nature with tasks of different integer values resulting in allocation and hence an energy reduction that is not exactly proportional across devices relative to their battery levels. Depending on the tasks assigned, some nodes may deplete their energy more than desired. Hence a periodic repartioning is necessary. However, due to the cost of executing an ILP, only a static initial allocation based on the ILP is utilized. Thus the DynA algorithms have a better chance at producing a longer system lifetime, due to their lower cost of execution and periodic task repartitioning.

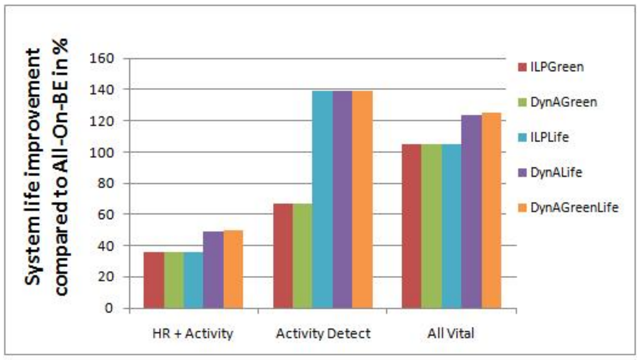

Figure 8.6: Percentage improvement in system battery lifetime achieved compared to processing all data on BE(*All-On-BE*

Fig. 8.7 and Fig. 8.8 show that *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* has up to 43% longer battery lifetime than *DynAGreen*. *DynAGreenLife* has slightly better system lifetime than *DynALife* for *All Vital* taskset. This is due to the fact that both algorithms attempt inexact solutions to the problem. *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 system energy and maximizing 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 targeting minimizing energy consumption; it has up to 150% more energy savings compared to *DynALife* and *DynAGreenLife*. When we compare *DynAGreenLife* and *DynALife*, both have similar system lifetime but *DynAGreenLife* consumes up to 40% less energy compared to *DynALife* algorithm.

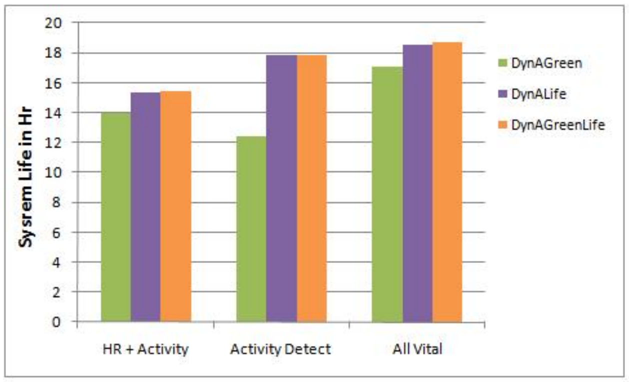

Figure 8.7: Comparison of system battery lifetime achieved by dynamic algorithms

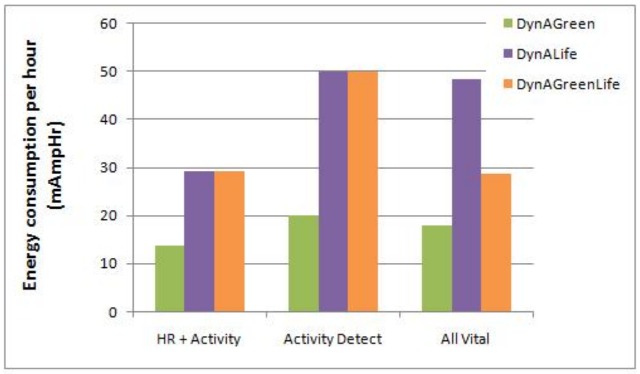

Figure 8.8: % Comparison of system lifetime achieved by dynamic algorithms

We show different task assignments given by ILPs and dynamic algorithms for *Activity Detect* taskset in Fig. 8.9 and 8.10. For this taskset *ILPGreen* and *DynAGreen* give same task assignment as shown in Fig. 8.9. Note that in this taskset we have 3 accelerometer sensing tasks which are pre-allocated to sensors and a logging task which is pre-allocated to BE. Both algorithms assign all the processing to LA as this assignment minimizes system energy. On the other hand, *ILPLife* and *DynALife* assign all three *Activity Count* tasks on their respective sensors as shown in Fig. 8.10. In our scenario LA has critically low battery level of 100mAh vs 300mAh of sensors. To extend the system lifetime, assignment shown in Fig. 8.10 reduces processing load on LA while increasing the load on *Sensors* even though assigning tasks on sensors results in higher overall system energy consumption. This achieves the objective of increasing system

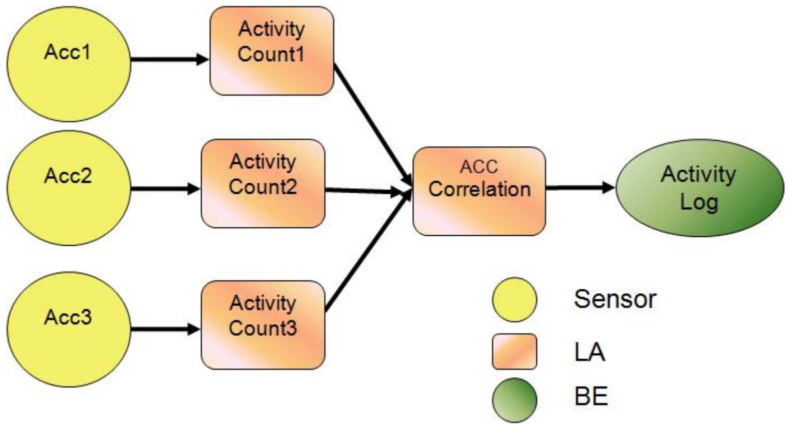

Figure 8.9: Task Assignment given by *ILPGreen* and *DynAGreen* algorithm

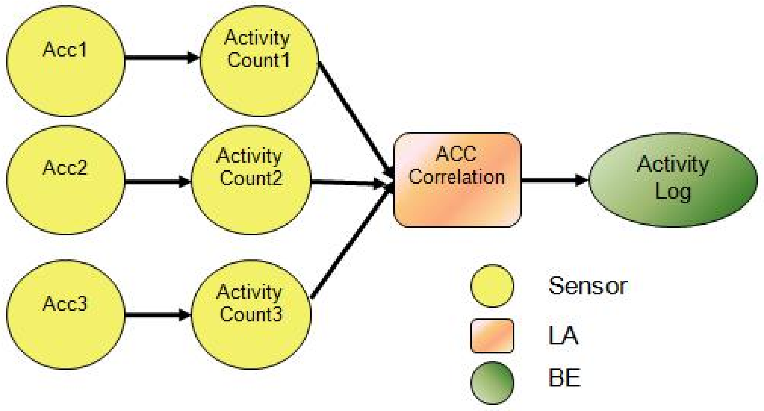

Figure 8.10: Task Assignment given by *ILPLife* and *DynALife* algorithm

lifetime at the cost of increased system energy. For this specific scenario, *ILPLife* and *DynALife* improve the system lifetime by 44% while consuming 148% more system energy compared to *ILPGreen* and *DynAGreen*. *DynAGreenLife* balances both system energy consumption and system lifetime.

8.3 Dynamic Adaptability

In the following set of experiments we simulate changes at runtime characteristics to demonstrate the capability of our dynamic algorithm to adapt. In this section we assume that the battery level is at 300 mAmp for each sensor and that the cellphone has a critically low battery level of 100 mAmp. To detect the changes in system characteristics, we measure each resource's radio link parameters such as transmit power and time spent in various states (receive, transmit, idle, sleep). These measurements are done over a period in which tasks execution is repeated. This period is the Lowest Common Multiple (LCM) of the period of all tasks. This window gives us a reproducible workload on each resource so we put a fair comparison of the various costs over different windows of time. For the *HR + Activity* and *Activity Detect* tasksets this window period is 1 minute and for *All Vital* taskset this window period is 2 minutes. We also use Infinite Impulse Response (IIR) filtering to smooth out measurements. This avoids the negative effect of any outlier to task assignment. We run our dynamic algorithms to recompute 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 implementation. Such a threshold helps reduce the frequency of algorithm execution and also avoids oscillations in assignment. We selected this threshold empirically for our simulation environment and envision that designer of a particular system can set this parameters based on observed task assignment oscillations.

To demonstrate dynamic adaptability of our algorithm, we put together two sets of experimental scenarios: 1) Illustrative scenarios with defined mobility, defined workload changes, only one base station and a smaller terrain area. These scenarios illustrate the effect of individual parameters such as radio link changes on system lifetime and system energy. 2) Urban scenarios with random mobility, multiple base station and larger terrain area to represent a more realistic situation. In all these scenarios, we show that our dynamic algorithms adapt to changing system parameters quickly and result in better performance compared to statically computed optimal solution obtained using ILPs.

8.3.1 Illustrative Scenarios

We have two main sources of changes in system parameters: 1) wireless channel condition changes which affect the communication energy parameters such as transmit and receive power. 2) workload changes which affect the computation energy parameters. In this section, we show affect of these two sources of change on system lifetime and system energy.

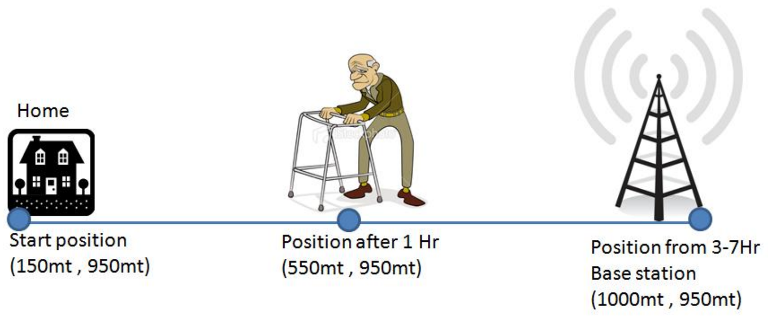

Figure 8.11: Experiment setup for illustrative scenarios

To estimate changes in link conditions we simulated a typical day of the user shown in Fig. 8.11. For this, we assume that the users home is away from the base station and thus wireless link conditions are relatively poor with lower effective data rates when the user is at home. The initial assignment is computed with the link conditions observed at home. 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 on the fly and updates the task assignments. The cost of UMTS transmission could increase by a factor of 10 depending on users distance from the base station.

Wireless Channel Condition Changes

Wireless channel conditions change over a period of time due to factors such as the mobility of the person using the system, weather conditions, and extra traffic on the system. When a user moves, wireless link conditions dynamically change. When a user is closer to a base-station, a higher-order modulation and coding scheme can be used with more bits/symbol transmitted and/or a higher code rate (less error correction) being utilized. This results in a more energy efficient wireless transmission for the

Table 8.6: Wireless channel condition changes:Experimental setup

Simulation time	% Execution time of tasks	Node position
0-30 Min	Initial	Near home
30 min-3 Hr	Initial	Moving towards base station
3-7 Hr	Initial	Near the base station
7-10 Hr	Initial	Moving towards home

Table 8.7: Improvement in system lifetime in case of change in wireless channel conditions

Algorithm	% Improvement in system lifetime relative to	HR + Activity	Activity Detection	All Vital
DynAGreen	ILPGreen	18.45	10.68	17.22
	ILPLife	18.45	-17.16	17.22
DynALife	ILPGreen	22.48	50	18.26
	ILPLife	22.48	12.27	18.26
DynAGreenLife	ILPGreen	23.79	24.88	22.18
	ILPLife	23.79	-6.53	22.14

same amount of information bits that need to be transmitted. Based on the higher energy efficiency and lower communication costs associated with WWAN transmissions, a dynamic task assignment may be chosen that could prefer communication over computation. Alternatively, as link conditions get worse when the user moves toward the edge of the cell, a lower order modulation and coding scheme may be used, resulting in a less energy efficient communication, which can lead to a dynamic task assignment that prefers computation over communication.

In Table. 8.7 we compare our three dynamic algorithms with static assignments given by *ILPGreen* and *ILPLife*. When user moves closer to the base station, our dynamic algorithms detect the reduced communication cost for *LA* to *BE* transmission and moves the processing tasks from LA to BE if that results in lower system energy or higher system lifetime. *DynAGreen* improves system lifetime for all tasksets compared to *ILPGreen* by 14% on an average. It even improves system lifetime compared to *ILPLife* taking advantage of new lower communication cost. Similarly *DynALife* improve system lifetime up to 50% compared to *ILPGreen* and *ILPLife*. Notice that *DynAGreenLife* has similar improvement as *DynALife* except for the case of *Activity Detection* taskset. In that scenario it improves the system lifetime compared to *ILPGreen* by 25% while reduces system lifetime compared to *ILPLife*. Reduction in system lifetime is compensated by reduced consumption of system energy as shown in Fig. 8.13. *DynAGreenLife* strikes a balance between in system energy and system lifetime as per the objective. *ILPGreen* and *ILPLife* generate‘ similar assignments for *HR+Activity* and *All Vital* tasksets given the same initial conditions.

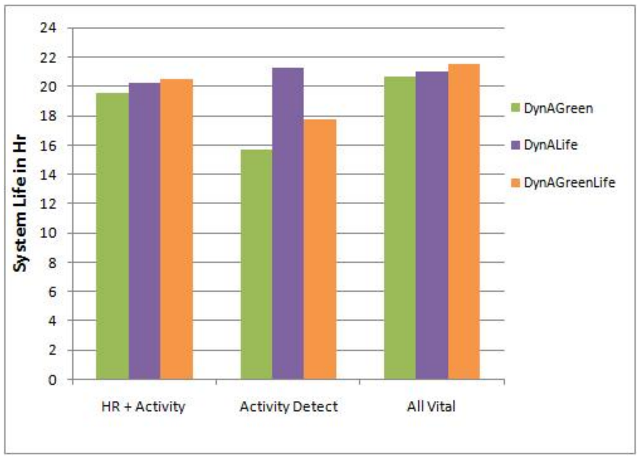

Figure 8.12: Change in wireless channel conditions: System lifetime comparison between dynamic algorithms

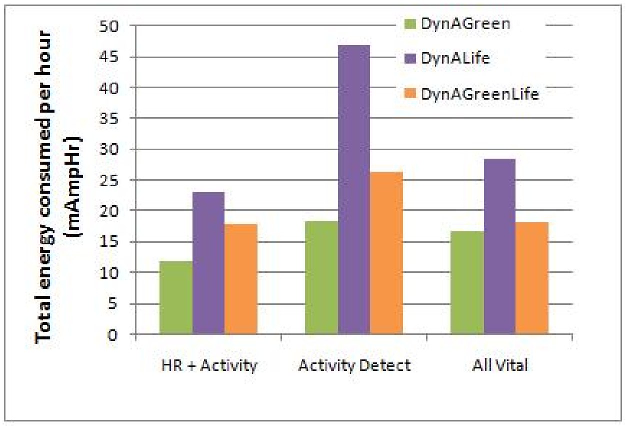

Figure 8.13: Change in wireless channel conditions: System energy consumption comparison

Fig. 8.12 and Fig. 8.13 show the effectiveness of *DynAGreenLife* in balancing increased system lifetime and reduced system energy. *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* taskset, *DynAGreen* assigns all processing tasks on *LA* when user is away from base station and it assigns all processing tasks on *BE* when user is closer to base station as shown in Fig. 8.14 . These assignments result in the lowest system energy. *DynALife* considers the very low battery charge of *LA* compared to sensors and assigns all single source processing tasks to respective sensors to reduce energy consumption on *LA*. *DynALife* keeps multi-source processing tasks on *LA* when *LA* is away from the base station and moves them to *BE* when *LA* gets closer to the base station as shown in Fig. 8.15. *DynALife* continously tries to reduce load on LA to extend its battery lifetime. *DynAGreenLife* strikes a balance between the two objectives. It generates the same assignment as *DynALife* when user is away from base station and it generates the same assignment as *DynAGreen* when user is closer to base station resulting in a better balance of system lifetime and system energy.

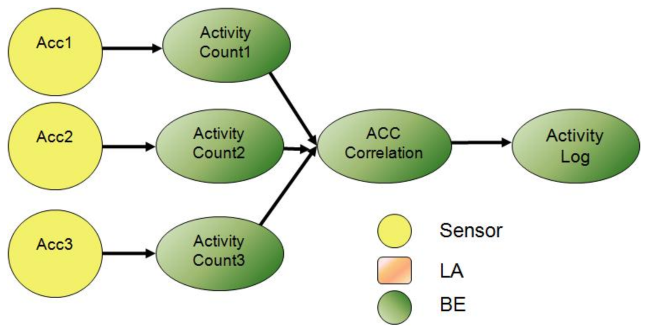

Figure 8.14: Assignment change given by *DynAGreen* when person is mobile and closer to the base station

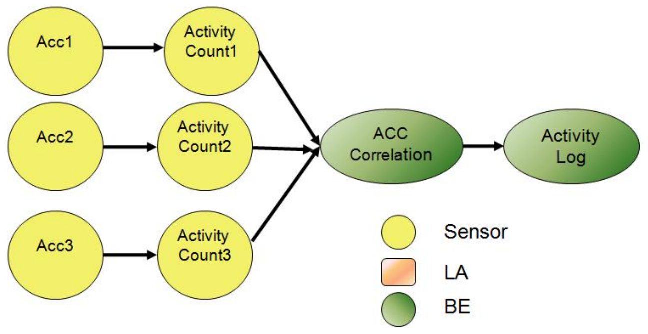

Figure 8.15: Assignment change given by *DynALife* when person is mobile and closer to the base station

Workload Changes

Dynamic load balancing is required in the system as the processing complexity of a task may change depending on amount and type of data, or due to increased load on a resource in the system such as the *LA*. When a patient is sedentary, the nature of the sensed information may be such that only coarse-grain processing of the information is sufficient. For example, if the sensed information is static or pseudo-static, the complexity of the tasks can be low based on prior processed information. Instead, if the sensed information varies significantly because of a change in the patients health conditions, or due to mobility, then the task complexity may increase. If a cell-phone is used as the *LA*, it may have additional processing to perform during the day for the user, in addition to supporting the health-care tasks. Thus, there can be an increased load due to increasing task complexity or due to other unrelated processing, which results in

Table 8.8: Improvement in system lifetime with dynamic load changes

Algorithm	% Improvement in system lifetime relative to	HR + Activity	Activity Detection	All Vital
DynAGreen	ILPGreen	9.19	0	0
	ILPLife	9.19	-18.60	0
DynALife	ILPGreen	12.99	41.01	10
	ILPLife	12.99	14.78	10
DynAGreenLife	ILPGreen	20.50	40.86	8.74
	ILPLife	20.50	14.66	8.74

additional energy utilization on the nodes. As a result of this varying load, a current task assignment may no longer be energy efficient from the overall system perspective, and dynamic load balancing is needed.

In our experiments we increased the execution time of processing tasks by a factor of 2 after 6 hours and set it to normal for next 6 hours to simulate two levels of processing load. We assume the same setup as the previous experiment except for the fact that LA is not mobile. Dynamic algorithms detect this variation in load and quickly recompute the new task assignment in response. Each tasks monitor their execution time and send a message to LA if they notice a significant change in their execution time. Also LA monitors its own CPU utilization and notifies dynamic algorithm if a significant change in utilization is detected. In our implementation we notify the algorithm on LA for any such small change but in practice a system designer can set a threshold for such notification.

In Table 8.8 we compare the percentage improvement in system lifetime obtained by all three dynamic algorithms relative to static assignments given by *ILPGreen* and *ILPLife*. In our scenario, when workload changes, our dynamic algorithms detect the increased computation cost and move the processing tasks from *LA* to *BE* if computation cost is higher than communication cost. *DynAGreen* improves system lifetime for *HR+Activity* taskset compared to *ILPGreen* by 10%. But the system lifetime improvement of *DynAGreen* over *ILPGreen* in *Activity Detection* and *All Vital* is null as they both give the same task assignment in these task graphs. Similarly *DynALife* improve system lifetime up to 51% compared to *ILPGreen* and *ILPLife*. Notice that *DynAGreenLife* has a similar improvement to *DynALife*. ILPGreen and ILPLife provide the same task assignment for *HR+Activity* and *All Vital* and hence improvements over *ILPGreen* and *ILPLife* for both the taskset are same. For *Activity Detection* task graph ILP solutions provide different task assignments, so dynamic algorithms provide different relative performance. Notice that all dynamic algorithms gains compared to *ILPGreen* are higher and gains compared to *ILPLife* are relatively lower. This is due to the fact that ILPLife is optimizing for system lifetime.

8.3.2 Urban Scenarios

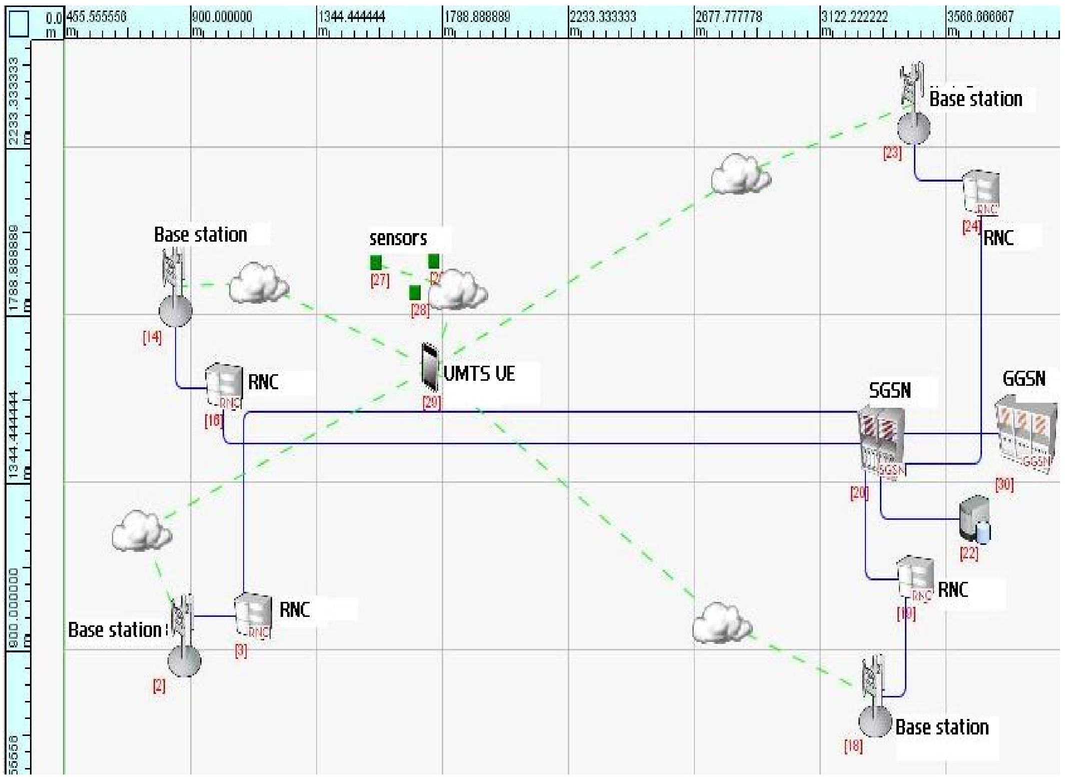

Figure 8.16: Simulation terrain

In this section, we show that our dynamic algorithms can handle real-life urban situation better then the ILP solutions. Near the edge of the cell handoffs may occur from one base-station to another. The link conditions at the edge are typically unfavorable regardless of the base-station that the device may be communicating with. After a handoff, if the user is closer to the center of a new base-station, then different task assignment may be chosen depending on the communication costs and the associated energy efficiency. To study the impact of such varying link conditions we have performed experiments with random mobility and multiple serving base-stations. For our experimental study, we created a 4900 meter x 4900 meter terrain with four base stations (Node B) connected to a UMTS-UE (*LA*) via a UMTS cellular network in Qualnet simulator as shown in Fig. 8.16. A UMTS network consist of three interacting domains; Core Network (CN), UMTS Terrestrial Radio Access Network (UTRAN) and User Equipment (UE). The main function of the core network is to provide switching, routing and transit for user traffic. Core network also contains the databases and network management functions. Base Station is referred as Node-B and control equipment for Node-B's is called Radio Network Controller (RNC) as shown in Fig. 8.16. Packet switched elements are Serving GPRS Support Node (SGSN) and Gateway GPRS Support Node (GGSN). 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. The experiment continues until one of the system components depletes its battery reserves. During the experiment the user pauses for a predefined duration (15 min or 45 min). This durations is chosen to simulate real time stops at park, grocery stores, pharmacy, etc.

We compare the system lifetime given by our *DynAGreenLife* algorithm to *ILPLife-Far*, *ILPLife-Near* and All-On-BE in Table 8.9. *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* by 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 by the ILP solution. These

Table 8.9: Improvement in system lifetime in case of random mobility

Taskgraph: HR + Activity				
Speed	Pause duration	% Improvement in system lifetime of *DynAGreenLife* relative to		
		ILPLife-Far	*ILPLife-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

Taskgraph: Activity Detection				
Speed	Pause duration	% Improvement in system lifetime of *DynAGreenLife* relative to		
		ILPLife-Far	*ILPLife-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

Taskgraph: All Vital				
Speed	Pause duration	% Improvement in system lifetime of *DynAGreenLife* relative to		
		ILPLife-Far	*ILPLife-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	8.28	46.73
Walk	45 min	34.88	7.36	56.59

two variations are used to cover ILP solution with best and worst communication cost between cell-phone and base station.

We monitor link conditions every 1 minute (which is LCM of all task periods) and run *DynAGreenLife* when the change in communication cost is 30% more than the last run of the algorithm. Table 8.9 shows that while a person is moving in 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 *ILPLife-Near*. In case of a person on a bike or walking, we observe even higher improvements compared to *ILPLife-Far*. User location trace for these experiments indicates that the user remains closer to the base station most of the time and hence improvements over *ILPLife-Near* is not as significant as the improvement of *ILPLife-Far*.

8.3.3 Utilizing Multiple ILP Solutions

Results in Table 8.9, suggest that a simple heuristic technique in which we switch between task assignments provided by *ILPLife-Near* and *ILPLife-Far* depending on user's proximity to a base station may achieve good results at lower online cost. This technique is referred as *ILP-Flipflop*. Such a solution would not handle other changes in the system such as changes in workload, addition of new tasks in taskset and varying channel conditions. To demonstrate this, we ran experiment with the same random mobility described in the previous section but also with changes in execution time of processing tasks every hour. Change in execution time represent a practical scenario in which a processing task might be asked to produce better quality result.

Fig. 8.17 shows results of such experiments for all three tasksets. We observe that in this scenario our *DynAGreenLife* algorithm is up to 30% and on an average 21% better than *ILP-Flipflop* technique described above. *ILP-Flipflop* is a binary solution and does not address intermediate channel conditions as well as workload changes. While the user is walking the improvement is higher for *HR+Activity* and *All Vital*. The reason for this behaviour is that the communication cost is in intermediate state (i.e. between

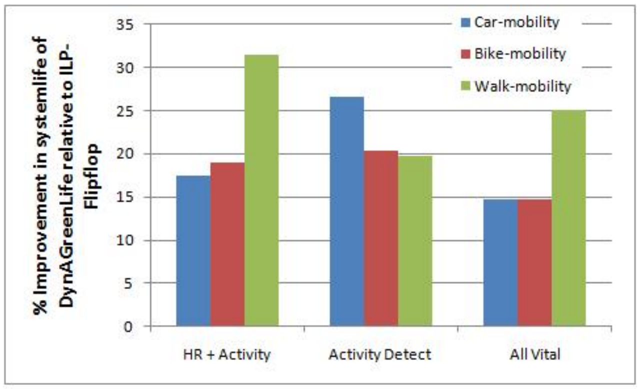

Figure 8.17: Percentage improvement in system lifetime achieved by *DynAGreenLife* relative to *ILP-FlipFlop* switching

best and worst conditions) for longer time as user is moving at a slower speed. For these two tasksets *DynAGreenLife*'s solution results in higher gains.

Chapter 9

Conclusion

Task assignment in a health care system consisting of heterogeneous resources could significantly impact the life and total energy consumption of the system. This thesis presents a number of static and dynamic task assignment strategies to save energy and extend system lifetime. Integer Linear Program(ILP) are formulated to generate a design time optimal task assignment to be used as a baseline for comparison. Given the dynamically changing nature of wireless systems and computationally expensive nature of *ILP*s, a heuristic algorithm(*DynALife*) and two dynamic graph-based partitioning algorithms(*DynAGreen and DynAGreenLife*) are proposed in this thesis. These three algorithms are computationally efficient and are able to adapt in real-time to changing system conditions. All these algorithms are implemented in the Qualnet discrete event wireless simulation environment to simulate various real time healthcare system scenarios. Task assignment generated by our dynamic algorithms performs similar to optimal task assignment of ILP for given static conditions. In the situation where cellphone has critically low battery and sensors are fully charged, our algorithms gives 1.4 times longer system lifetime compared to processing all data on backend server. *DynAGreenLife* has up to 43% longer battery life compared to *DynAGreen* algorithm and consumes up to 40% less energy than *DynALife* algorithm. Thus, *DynAGreenLife* algorithm balances both system lifetime and energy.

In other experimental results, which are based on real-life examples of wireless healthcare networks, dynamic algorithms outperforms the ILP-based solution by up to 50% due to variation in channel conditions and by up to 41% due to variation in load. *DynAGreenLife* algorithm has on an average 21% longer system life than ILP-flipflop technique. Thus, all this experiments show that our proposed dynamic algorithms performs similar to optimal solution given by ILP in static conditions and significantly outperforms the ILP based solution in dynamically changing conditions by changing the task assignments at runtime.

Appendices

Node placement files

This section gives the details of the node placement of the taskgraph as used in the code. The format used my Qualnet for .nodes file is:
(Node#) (time) (x,y,z) (azimuth) (elevation)
In all the files below, Node 1 corresponds to Local Aggregator Node 2 corresponds to Node B(base station) Node 3 corresponds to RNC Node 4 corresponds to SGSN Node 5 corresponds to GGSN Node 6 corresponds to HLR Node 7 corresponds to Backend server Node 8 corresponds to ECG sensor Node 9 corresponds to ACC1 sensor Node 10 corresponds to ACC2 sensor Node 11 corresponds to ACC3 sensor Node 12 corresponds to BP sensor

Static Conditions

Node placement file for HR + Activity:

```
1 0 (150.0, 950.0, 0.0) 0 0
2 0 (1000.08, 900.42, 0.0) 0 0
3 0 (1119.35, 984.96, 0.0) 0 0
4 0 (1331.72, 967.78, 0.0) 0 0
5 0 (1200.65, 760.01, 0.0) 0 0
6 0 (1455.78, 736.62, 0.0) 0 0
7 0 (1244.84, 553.2, 0.0) 0 0
```

```
8 0 (149, 950, 0.0) 0 0
9 0 (150.0, 951.0, 0.0) 0 0
```

Node placement file for Activity Detect:

```
1 0 (150.0, 950.0, 0.0) 0 0
2 0 (1000.08, 900.42, 0.0) 0 0
3 0 (1119.35, 984.96, 0.0) 0 0
4 0 (1331.72, 967.78, 0.0) 0 0
5 0 (1200.65, 760.01, 0.0) 0 0
6 0 (1455.78, 736.62, 0.0) 0 0
7 0 (1244.84, 553.2, 0.0) 0 0
9 0 (150.0, 951.0, 0.0) 0 0
10 0 (151.0, 950.0, 0.0) 0 0
11 0 (149.0, 950.0, 0.0) 0 0
```

Node placement file for All-Vital:

```
1 0 (150.00, 950.00, 0.0) 0 0
2 0 (1000.08, 950.00, 0.0) 0 0
3 0 (1119.35, 984.96, 0.0) 0 0
4 0 (1331.72, 967.78, 0.0) 0 0
5 0 (1200.65, 760.01, 0.0) 0 0
6 0 (1455.78, 736.62, 0.0) 0 0
7 0 (1244.84, 553.2, 0.0) 0 0
8 0 (150.00, 951.00, 0.0) 0 0
9 0 (149.00, 950.00, 0.0) 0 0
10 0 (150.00, 949.00, 0.0) 0 0
11 0 (151.00, 950.00, 0.0) 0 0
12 0 (149.00, 951.00, 0.0) 0 0
```

Wireless Channel Condition Changes

Node placement file for HR + Activity :

```
3 0 (1059.35, 984.96, 0.0) 0 0
4 0 (1331.72, 967.78, 0.0) 0 0
5 0 (1200.65, 760.01, 0.0) 0 0
6 0 (1455.78, 736.62, 0.0) 0 0
7 0 (1244.84, 553.2, 0.0) 0 0
2 0 (1005.08, 950.00, 0.0) 0 0
13 0 (2000, 953,0) 0 0
14 0 (2010, 959,0) 0 0

1 0 (50.00, 950.00, 0.0) 0 0
1 10M (300.00, 950.00, 0.0) 0 0
1 20M (775.00, 950.00, 0.0) 0 0
1 40M (1000.00, 950.00, 0.0) 0 0
1 100M (1000.00, 950.00, 0.0) 0 0
1 120M (1300.00, 950.00, 0.0) 0 0
1 130M (1500.00, 950.00, 0.0) 0 0
1 150M (1750.00, 950.00, 0.0) 0 0
1 160M (2000.00, 950.00, 0.0) 0 0
1 170M (1750.00, 950.00, 0.0) 0 0
1 190M (1500.00, 950.00, 0.0) 0 0
1 200M (1300.00, 950.00, 0.0) 0 0
1 220M (1000.00, 950.00, 0.0) 0 0
1 230M (775.00, 950.00, 0.0) 0 0
1 250M (500.00, 950.00, 0.0) 0 0
1 260M (300.00, 950.00, 0.0) 0 0
1 280M (50.00, 950.00, 0.0) 0 0

8 0 (50.00, 951.00, 0.0) 0 0
```

```
8 10M (300.00, 951.00, 0.0) 0 0
8 20M (775.00, 951.00, 0.0) 0 0
8 40M (1000.00, 951.00, 0.0) 0 0
8 100M (1000.00, 951.00, 0.0) 0 0
8 120M (1300.00, 951.00, 0.0) 0 0
8 130M (1500.00, 951.00, 0.0) 0 0
8 150M (1750.00, 951.00, 0.0) 0 0
8 160M (2000.00, 951.00, 0.0) 0 0
8 170M (1750.00, 951.00, 0.0) 0 0
8 190M (1500.00, 951.00, 0.0) 0 0
8 200M (1300.00, 951.00, 0.0) 0 0
8 220M (1000.00, 951.00, 0.0) 0 0
8 230M (775.00, 951.00, 0.0) 0 0
8 250M (500.00, 951.00, 0.0) 0 0
8 260M (300.00, 951.00, 0.0) 0 0
8 280M (50.00, 951.00, 0.0) 0 0

9 0 (50.00, 950.50, 0.0) 0 0
9 10M (300.00, 950.50, 0.0) 0 0
9 20M (775.00, 950.50, 0.0) 0 0
9 40M (1000.00, 950.50, 0.0) 0 0
9 100M (1000.00, 950.50, 0.0) 0 0
9 120M (1300.00, 950.50, 0.0) 0 0
9 130M (1500.00, 950.50, 0.0) 0 0
9 150M (1750.00, 950.50, 0.0) 0 0
9 160M (2000.00, 950.50, 0.0) 0 0
9 170M (1750.00, 950.50, 0.0) 0 0
9 190M (1500.00, 950.50, 0.0) 0 0
9 200M (1300.00, 950.50, 0.0) 0 0
9 220M (1000.00, 950.50, 0.0) 0 0
```

```
9 230M (775.00, 950.50, 0.0) 0 0

9 250M (500.00, 950.50, 0.0) 0 0

9 260M (300.00, 950.50, 0.0) 0 0

9 280M (50.00, 950.50, 0.0) 0 0
```

Node placement file for Activity Detection:

```
3 0 (1119.35, 984.96, 0.0) 0 0

4 0 (1331.72, 967.78, 0.0) 0 0

5 0 (1200.65, 760.01, 0.0) 0 0

6 0 (1455.78, 736.62, 0.0) 0 0

7 0 (1244.84, 553.2, 0.0) 0 0

2 0 (1000.08, 950.00, 0.0) 0 0


1 0 (50.00, 950.00, 0.0) 0 0

1 1H (500.00, 950.00, 0.0) 0 0

1 2H (775.00, 950.00, 0.0) 0 0

1 4H (1000.00, 950.00, 0.0) 0 0

1 8H (1000.00, 950.00, 0.0) 0 0

1 10H (150.00, 950.00, 0.0) 0 0

1 11H (50.00, 950.00, 0.0) 0 0

1 12H (500.00, 950.00, 0.0) 0 0

1 13H (775.00, 950.00, 0.0) 0 0

1 14H (1000.00, 950.00, 0.0) 0 0

1 18H (1000.00, 950.00, 0.0) 0 0

1 20H (150.00, 950.00, 0.0) 0 0

1 21H (50.00, 950.00, 0.0) 0 0


9 0 (49.00, 950.00, 0.0) 0 0

9 1H (499.00, 950.00, 0.0) 0 0

9 2H (774.00, 950.00, 0.0) 0 0
```

```
9 4H (999.00, 950.00, 0.0) 0 0
9 8H (999.00, 950.00, 0.0) 0 0
9 10H (149.00, 950.00, 0.0) 0 0
9 11H (49.00, 950.00, 0.0) 0 0
9 12H (499.00, 950.00, 0.0) 0 0
9 13H (774.00, 950.00, 0.0) 0 0
9 14H (999.00, 950.00, 0.0) 0 0
9 18H (999.00, 950.00, 0.0) 0 0
9 20H (149.00, 950.00, 0.0) 0 0
9 21H (49.00, 950.00, 0.0) 0 0

10 0 (50.00, 951.00, 0.0) 0 0
10 1H (500.00, 951.00, 0.0) 0 0
10 2H (775, 951.00, 0.0) 0 0
10 4H (1000.00, 951.00, 0.0) 0 0
10 8H (1000.00, 951.00, 0.0) 0 0
10 10H (150.00, 951.00, 0.0) 0 0
10 11H (50.00, 951.00, 0.0) 0 0
10 12H (500.00, 951.00, 0.0) 0 0
10 13H (775, 951.00, 0.0) 0 0
10 14H (1000.00, 951.00, 0.0) 0 0
10 18H (1000.00, 951.00, 0.0) 0 0
10 20H (150.00, 951.00, 0.0) 0 0
10 21H (50.00, 951.00, 0.0) 0 0

11 0 (51.00, 950.00, 0.0) 0 0
11 1H (501.00, 950.00, 0.0) 0 0
11 2H (776.00, 950.00, 0.0) 0 0
11 4H (1001.00, 950.00, 0.0) 0 0
11 8H (1001.00, 950.00, 0.0) 0 0
```

```
11 10H (151.00, 950.00, 0.0) 0 0
11 11H (51.00, 950.00, 0.0) 0 0
11 12H (501.00, 950.00, 0.0) 0 0
11 13H (776.00, 950.00, 0.0) 0 0
11 14H (1001.00, 950.00, 0.0) 0 0
11 18H (1001.00, 950.00, 0.0) 0 0
11 20H (151.00, 950.00, 0.0) 0 0
11 21H (51.00, 950.00, 0.0) 0 0
```

Node placement file for All-Vital:

```
3 0 (1119.35, 984.96, 0.0) 0 0
4 0 (1331.72, 967.78, 0.0) 0 0
5 0 (1200.65, 760.01, 0.0) 0 0
6 0 (1455.78, 736.62, 0.0) 0 0
7 0 (1244.84, 553.2, 0.0) 0 0
2 0 (1000.08, 950.00, 0.0) 0 0

1 0 (50.00, 950.00, 0.0) 0 0
1 1H (500.00, 950.00, 0.0) 0 0
1 2H (775.00, 950.00, 0.0) 0 0
1 4H (1000.00, 950.00, 0.0) 0 0
1 8H (1000.00, 950.00, 0.0) 0 0
1 10H (150.00, 950.00, 0.0) 0 0
1 11H (50.00, 950.00, 0.0) 0 0
1 12H (500.00, 950.00, 0.0) 0 0
1 13H (775.00, 950.00, 0.0) 0 0
1 15H (1000.00, 950.00, 0.0) 0 0
1 19H (1000.00, 950.00, 0.0) 0 0
1 21H (150.00, 950.00, 0.0) 0 0
1 22H (50.00, 950.00, 0.0) 0 0
```

```
8 0 (49.00, 951.00, 0.0) 0 0
8 1H (499.00, 951.00, 0.0) 0 0
8 2H (774.00, 951.00, 0.0) 0 0
8 4H (999.00, 951.00, 0.0) 0 0
8 8H (999.00, 951.00, 0.0) 0 0
8 10H (149.00, 951.00, 0.0) 0 0
8 11H (49.00, 951.00, 0.0) 0 0
8 12H (499.00, 951.00, 0.0) 0 0
8 13H (774.00, 951.00, 0.0) 0 0
8 15H (999.00, 951.00, 0.0) 0 0
8 19H (999.00, 951.00, 0.0) 0 0
8 21H (149.00, 951.00, 0.0) 0 0
8 22H (49.00, 951.00, 0.0) 0 0

9 0 (49.00, 950.00, 0.0) 0 0
9 1H (449.00, 950.00, 0.0) 0 0
9 2H (774.00, 950.00, 0.0) 0 0
9 4H (999.00, 950.00, 0.0) 0 0
9 8H (999.00, 950.00, 0.0) 0 0
9 10H (149.00, 950.00, 0.0) 0 0
9 11H (49.00, 950.00, 0.0) 0 0
9 12H (499.00, 950.00, 0.0) 0 0
9 13H (774.00, 950.00, 0.0) 0 0
9 15H (999.00, 950.00, 0.0) 0 0
9 19H (999.00, 950.00, 0.0) 0 0
9 21H (149.00, 950.00, 0.0) 0 0
9 22H (49.00, 950.00, 0.0) 0 0

10 0 (50.00, 951.00, 0.0) 0 0
```

```
10 1H (500.00, 951.00, 0.0) 0 0

10 2H (775, 951.00, 0.0) 0 0

10 4H (1000.00, 951.00, 0.0) 0 0

10 8H (1000.00, 951.00, 0.0) 0 0

10 10H (150.00, 951.00, 0.0) 0 0

10 11H (50.00, 951.00, 0.0) 0 0

10 12H (500.00, 951.00, 0.0) 0 0

10 13H (775.00, 951.00, 0.0) 0 0

10 15H (1000.00, 951.00, 0.0) 0 0

10 19H (1000.00, 951.00, 0.0) 0 0

10 21H (150.00, 951.00, 0.0) 0 0

10 22H (50.00, 951.00, 0.0) 0 0



11 0 (51.00, 950.00, 0.0) 0 0

11 1H (501.00, 950.00, 0.0) 0 0

11 2H (776.00, 950.00, 0.0) 0 0

11 4H (1001.00, 950.00, 0.0) 0 0

11 8H (1001.00, 950.00, 0.0) 0 0

11 10H (151.00, 950.00, 0.0) 0 0

11 11H (51.00, 950.00, 0.0) 0 0

11 12H (501.00, 950.00, 0.0) 0 0

11 13H (776.00, 950.00, 0.0) 0 0

11 15H (1001.00, 950.00, 0.0) 0 0

11 19H (1001.00, 950.00, 0.0) 0 0

11 21H (151.00, 950.00, 0.0) 0 0

11 22H (51.00, 950.00, 0.0) 0 0

12 0 (49.00, 949.00, 0.0) 0 0

12 1H (499.00, 949.00, 0.0) 0 0
```

```
12 2H (774.00, 949.00, 0.0) 0 0
12 4H (999.00, 949.00, 0.0) 0 0
12 8H (999.00, 949.00, 0.0) 0 0
12 10H (149.00, 949.00, 0.0) 0 0
12 11H (49.00, 949.00, 0.0) 0 0
12 12H (499.00, 949.00, 0.0) 0 0
12 13H (774.00, 949.00, 0.0) 0 0
12 15H (999.00, 949.00, 0.0) 0 0
12 19H (999.00, 949.00, 0.0) 0 0
12 21H (149.00, 949.00, 0.0) 0 0
12 22H (49.00, 949.00, 0.0) 0 0
```

References

[1] Aghajan, H., Augusto, J. C., Mccullagh, P., and ann Walkden, J., 2007: Distributed vision-based accident management for assisted living. In *In Int. Conf. on Smart homes and health Telematics (ICOST*.

[2] Au, L., Wu, W., Batalin, M., Mclntire, D., and Kaiser, W., 2007: Microleap: Energy-aware wireless sensor platform for biomedical sensing applications. 158 –162. doi:10.1109/BIOCAS.2007.4463333.

[3] A.V.Halteren, K. W. D. K., R. Bults, and Windya, I., 2004: Mobile patient monitoring: The mobihealth system.

[4] Bokhari, S., 1988: Partitioning problems in parallel, pipeline, and distributed computing. *Computers, IEEE Transactions on*, **37**(1), 48 –57. ISSN 0018-9340. doi: 10.1109/12.75137.

[5] Boykov, Y., and Kolmogorov, V., 2004: An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **26**(9), 1124 –1137. ISSN 0162-8828.

[6] Cardionet, 2008: http://www.cardionet.com/.

[7] CWPHS, P., 2008: http://cwphs.calit2.net/index.php?option=commid=79.

[8] Dabiri, F., Vahdatpour, A., Noshadi, H., and Hagopian, H. e., 2008: Ubiquitous personal assistive system for neuropathy. In *HealthNet '08: Proceedings of the 2nd International Workshop on Systems and Networking Support for Health Care and Assisted Living Environments*, 1–6. ISBN 978-1-60558-199-6.

[9] Jeannot, E., and Knutsson, B., 2002: Adaptive online data compression. 379 – 388. ISSN 1082-8907. doi:10.1109/HPDC.2002.1029938.

[10] K. Patrick, M. A. L. D. M. Z. C. R. W. G. e., F. Raab, 2009: A text messagebased intervention for weight loss: Randomized controlled trial.

[11] Lp-solve., 2008: http://lpsolve.sourceforge.net/5.5/.

[12] Nahapetian, A., Dabiri, F., and Sarrafzadeh, M., 2006: Energy minimization and reliability for wearable medical applications. 8 pp. –318. doi:10.1109/ICPPW. 2006.36.

[13] P. Aghera, D. F. A. C. T. R., D. Krishnaswamy, 2010: Dynaheal: Dynamic energy efficient task assignment for wireless healthcare systems.

[14] P. Aghera, T. R., D. Krishnaswamy, 2010: Dynagreen: Hierarchical dynamic energy efficient task assignment for wireless healthcare systems.

[15] Scalable Network Technologies, I., 2004: Qualnet simulator.

[16] Stone, H., 1977: Multiprocessor scheduling with the aid of network flow algorithms. *Software Engineering, IEEE Transactions on*, **SE-3**(1), 85 – 93. ISSN 0098-5589.

[17] Tian, Y., and Ekici, E., 2007: Cross-layer collaborative in-network processing in multihop wireless sensor networks. *Mobile Computing, IEEE Transactions on*, **6**(3), 297 –310. ISSN 1536-1233. doi:10.1109/TMC.2007.39.

[18] Tian, Y., Ekici, E., and Ozguner, F., 2005: Energy-constrained task mapping and scheduling in wireless sensor networks. 8 pp. –218. doi:10.1109/MAHSS.2005. 1542802.

[19] Wang, Q., Shin, W., Liu, X., Zeng, Z., Oh, C., AlShebli, B., Caccamo, M., Gunter, C., Gunter, E., Hou, J., Karahalios, K., and Sha, L., 2006: I-living: An open system architecture for assisted living. volume 5, 4268 –4275. doi:10.1109/ICSMC.2006. 384805.

[20] Wood, A., Virone, G., Doan, T., Cao, Q., Selavo, L., Wu, Y., Fang, L., He, Z., Lin, S., and Stankovic, J., 2006: Alarm-net: Wireless sensor networks for assisted-living and residential monitoring. Technical report.

[21] Wu, W., Au, L., Jordan, B., Stathopoulos, T., Batalin, M., Kaiser, W., Vahdatpour, A., Sarrafzadeh, M., Fang, M., and Chodosh, J., 2008: The smartcane system: an assistive device for geriatrics. In *BodyNets '08: Proceedings of the ICST 3rd international conference on Body area networks*, 1–4. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium. ISBN 978-963-9799-17-2.

[22] Xu, R., Li, Z., Wang, C., and Ni, P., 2003: Impact of data compression on energy consumption of wireless-networked handheld devices. In *in Proceedings of the 23rd IEEE International Conference on Distributed Computing Systems (ICDCS03*, 302–311. IEEE Computer Society.

[23] Yu, Y., and Prasanna, V. K., 2005: Energy-balanced task allocation for collaborative processing in wireless sensor networks. volume 10, 115–131. ISSN 1383-469X.