Burst Communication by Means of Buffer Allocation in Body Sensor Networks: Exploiting Signal Processing to Reduce the Number of Transmissions

Hassan Ghasemzadeh, *Student Member, IEEE,* Vitali Loseu, *Student Member, IEEE,*
Sarah Ostadabbas, *Student Member, IEEE,* and Roozbeh Jafari, *Member, IEEE*

***Abstract*—Monitoring human movements using wireless sensory devices promises to revolutionize the delivery of healthcare services. Such platforms use inertial information of their subjects for motion analysis. Potentially, each action or disease can be discovered by collaborative processing of sensor data from multiple locations on the body. This functionality is provided by a Body Sensor Network (BSN), which consists of several wireless sensor nodes positioned on different parts of the body. In spite of the revolutionary potential of this platform, power requirements and wearability have limited the commercialization of these systems. In this paper, we present an energy-efficient communication model for BSN applications which uses buffers to limit communication to short bursts, decreasing power usage and simplifying the communication. We formulate an optimization problem to reduce transmissions among sensor nodes and present an ILP-based solution and a fast greedy heuristic algorithm. We show that despite the decreased transmission efficiency, our greedy algorithm can be adopted for fast allocation of buffers in real-time. We experimentally compare the performance of both of the proposed approaches to the performance of an unbuffered system. Our results demonstrate that ILP and greedy solutions can reduce the amount of transmissions by an average factor of 70 and 41, respectively.**

***Index Terms*—Body Sensor Networks, Collaborative Signal Processing, Burst Transmission, Buffer Allocation.**

I. Introduction

TECHNOLOGICAL advances in wireless communication, sensor design and embedded processors have led to the development of pervasive Body Sensor Networks (BSNs) that enable wearable and mobile healthcare monitoring systems. Such platforms consist of a set of miniaturized sensor nodes which sense physiological and environmental data, initiate actions and trigger alarms during an emergency. BSNs can be used for remote patient monitoring and testing which enables a shift of healthcare from a traditional clinical setting to the home. This shift will reduce costs, allow collection of information previously unavailable, and give patients more control over their care.

Despite the variety of potential applications, proliferation of continuous health monitoring is still limited due to application-dependent platform constraints in terms of computation, communication, battery lifetime and storage. Often the problem is over-constrained, and so the designer must relax some constraints to produce a solution. Consequently, techniques improving the efficiency of resource usage can greatly expand the efficiency of the platform.

In BSNs, energy is constrained to a small battery of each sensor node. One of the biggest consumers of energy is communication [1]. In traditional wireless sensor networks, a significant amount of energy can be saved by improving the routing protocol. However, on a BSN, sensor nodes are quite proximate, which results in a single-hop network. From the application perspective communication cost depends only on the amount of data to be sent. Because of this, energy saving techniques frequently focus on decreasing the amount of data that needs to be communicated through greater local processing [2]. There is still advantage to be gained by application-aware, data-agnostic communication optimizations. In many applications, channel usage is fragmented—that is communication is frequent, but sparse.

In this paper, we propose a method for using buffers to transmit in bursts. Burst transmission can reduce energy per bit [3], and can simplify communication by lowering packet scheduling overhead [4]. Our technique assigns each signal processing module a large buffer which is sized to minimize communication. Since many medical monitoring applications require only periodic reports, real-time results are unnecessary [5]. Removing immediate data-processing deadlines permits an interesting optimization based on burst transmissions. We model signal processing as a directed acyclic dependency graph in which processing modules use results from other nodes for data fusion. The dependency graphs are used to introduce an IP (Integer Programming) of our optimization problem with a separable convex function. A polynomial-time greedy algorithm is then proposed to compensate computational complexity of the ILP-based solution for dynamic allocation of buffers in real-time.

II. Related Work

Several authors have addressed the energy limitations of BSNs with respect to signal processing needs. Authors in [6] pose optimization problems for minimizing the number of active nodes and for maximizing system lifetime while maintaining high classification accuracy for action recognition. Similarly, a task graph can be constructed using timing constraints and data dependencies [7]. Using the task dependency

Manuscript received 1 May 2009; revised 16 February 2010.

The authors are with the Department of Electrical Engineering, University of Texas at Dallas, Richardson, Texas 75080-3021, USA (e-mail: h.ghasemzadeh@utdallas.edu.)

Digital Object Identifier 10.1109/JSAC.2010.100912.

graph, the system can determine the critical paths, which can be used in development of a scheduling mechanism to distribute the unused time (slacks) among tasks such that the overall energy cost is minimized. The decision tree model in [8] is used incremental movement classification, where only informative nodes are involved in each classification decision. The node activation policy in [9] activates sensor nodes dynamically according to the local observations made by individual nodes.

Burst transmission requires buffers to accumulate results between transmissions [10]. The concept of buffers has been utilized in several domains in order to optimize communication system. It has been traditionally used in ATM networks to improve system throughput by allowing communication parties to dynamically allocate their buffers to multiple nodes [11]. In networks-on-chip (NoC), efficient distribution of buffers among input channels of on-chip routers can significantly increase overall performance of the system [12]. Buffers have been used in embedded software to minimize memory requirements of runtime software components [13]. In real-time distributed systems, buffers have been employed to avoid communication delay among synchronous processes and to preserve data semantics [14].

Nevertheless, to the best of our knowledge, the issue of communication minimization by means of buffers in BSNs has not been investigated previously. In our work, we explore a buffer technique for BSN applications with relaxed time deadlines. We first outline an ILP formulation of this problem aiming to maintain optimal size buffers across the network. Although the formulation can be used for static allocation of buffers, computational complexity of the ILP limits its utilization in real-time scenarios. Inspired by this need, we further present a greedy approach for buffer assignment.

III. Preliminaries

Before formulating the problem of buffer allocation in BSNs, several concepts must be presented, including a more thorough description of the platform, signal processing model, and definition of dependency graphs in the context of this problem.

A. System Architecture and Signal Processing

Our system consists of several sensor nodes, each equipped with a processing module and a custom-designed sensor board powered by a Li-Ion battery. The processing module is a mote with embedded radio for communication. Each sensor board has a tri-axial accelerometer and a bi-axial gyroscope. The system aims to detect physical movements of the subject wearing the system. A number of sensor nodes are placed on different joints of the human body. Inertial data obtained by each sensor node is subject to physical action recognition and further processing.

A typical system of physical activity recognition intends to classify a set of movements of interest. Pattern classification techniques are mostly employed to distinguish each action from the rest. Our per-node signal processing involves several modules including *preprocessing*, *segmentation*, *feature extraction*, and *classification* [15]. In preprocessing phase, the data is collected from each sensor node and is passed through a filter to reduce high frequency noise. During the segmentation the signal is divided into segments that represent complete actions. This is done by annotating the start and end of each movement. A feature extraction block calculates a set of statistical and morphological features from each signal segment. Each sensor node makes a local classification decision on the movement being performed by the subject. A final decision based on the all local results is made by a central node.

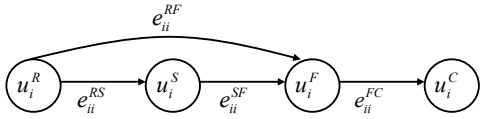

Fig. 1. **Intra-node dependency subgraph**

B. Collaborative Model

The signal processing model described in Section III-A is usually implemented in a distributed manner over a BSN. We represent information flow across the network with our dependency graph model. In a system of n sensor nodes, dependency graph is composed of n subgraphs connected through inter-node links. Each subgraph represents static information dependencies within a node as shown in Fig. 1 where u_i^R, u_i^S, u_i^F and u_i^C denote sensor reading and preprocessing, segmentation, feature extraction and classification blocks respectively. As mentioned previously, features are extracted by locating the annotation points in the filtered signal and calculating a set of statistical and morphological features from the resulting segment.

***Definition** 1:* Given a set of n sensor nodes $s_1, \ldots, s_n$, *intra-node dependency subgraph* for node s_i is defined by $G_i = (V_i, E_i)$ where V_i is the set of four vertices and E_i is the set of four edges. Each vertex, denoted by u_i^μ, corresponds to a processing unit, and each edge is denoted by $e_{ii}^{\mu\eta}$ $(\mu, \eta \in \{R, S, F, C\})$ representing intra-node dependencies.

Inter-node links which represent dependencies across sensor nodes are used to connect subgraphs and form a dependency graph. An inter-node link is determined according to dependencies induced by the application of interest.

***Definition** 2:* Given a set of n sensor nodes $s_1, \ldots, s_n$ and inter-node dependencies, *dependency graph* $G = (V, E)$ is formed by connecting n intra-node dependency subgraphs $G_1, \ldots, G_n$ through dependency links E_b defined by application criteria. The set of edges E is given by

$$E = E_w \cup E_b \tag{1}$$

where the set of intra-node edges E_w is given by

$$E_w = \bigcup_{i=1}^{n} E_i \qquad i = 1, ..., n \tag{2}$$

and the set of vertices V is defined by

$$V = \bigcup_{i=1}^{n} V_i \qquad i = 1, ..., n \tag{3}$$

where each V_i is the set of vertices within subgraph G_i.

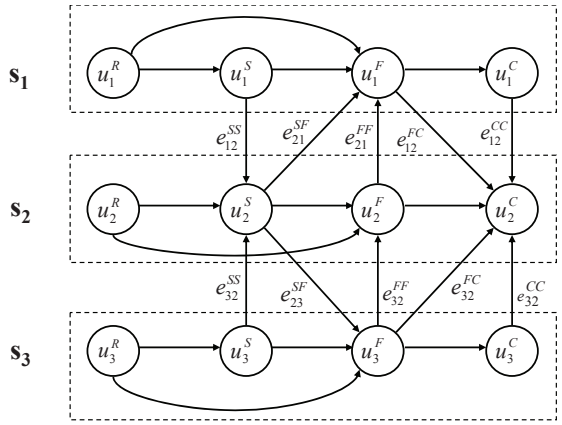

Fig. 2. **An example of dependency graph for a three-node network**

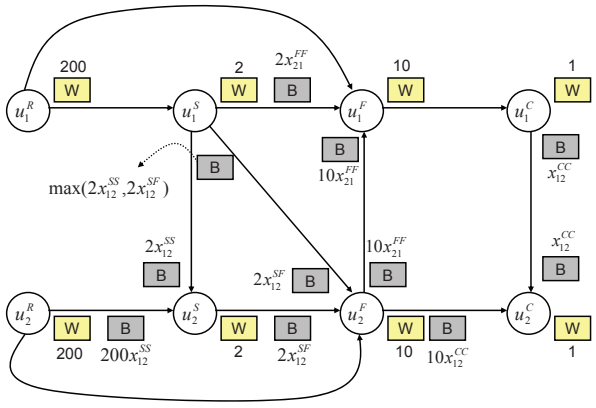

Fig. 3. **Example of buffer assignment method**

As an abstract example, Fig. 2 shows a dependency graph for a network of three sensor nodes where processing units are connected through 10 inter-node dependency links. Intra-node subgraphs are highlighted by the dashed boxes.

In our activity monitoring system, practical inter-node links $e_{ij}^{\mu\eta}$ connecting the processing block μ at node s_i to the processing block η at s_j are presented as follows. A link e_{ij}^{SS} shows that the segmentation block at sensor s_j requires segmentation results from sensor s_i. A lack of prominent patterns prevents s_j from properly determining start and end of actions. When a subject is walking, a node placed on the leg observes high quality pattern, but a head node observes irrelevant patterns. A link e_{ij}^{SF} is required when node s_i becomes a master node for segmentation and provides node s_j with the annotation points (fragment of sensor data stream signifying a certain activity of predefined class type) for feature extraction. When s_j requires information on features calculated by s_i to extract features, a link e_{ij}^{FF} is added. If classification at node s_j is contingent on certain features from s_i, a link e_{ij}^{FC} is required. A segmentation unit may need information on current movement detected by another node. In this case, a link e_{ij}^{CS} is added to the dependency graph. A link e_{ij}^{CF} is required if the feature extraction depends on classification results provided by other nodes. When sensor node s_j is considered as classifier aggregator, links of the form e_{ij}^{CC} are considered allowing other nodes to transmit local classification to a central node.

IV. Problem Definition

The idea behind using buffers is to transmit the maximum amount of data in short time intervals. The large number of data blocks produced by each processing unit is stored locally and is transmitted using available bandwidth. This would conform to real situations of healthcare systems where physicians are interested in receiving reports on daily activities rather than immediate reports. By assuming no immediate deadlines in the system, we maintain individual buffers on each link and transmit the data blocks separately for each inter-node link.

***Problem** 1:* Given dependency graph G, each inter-node link $e_{ij}^{\mu\eta}$ is associated with a number $x_{ij}^{\mu\eta}$ denoting the number of actions for which data blocks produced by the source unit u_i^μ are buffered prior to every transmission. The objective is to find values $x_{ij}^{\mu\eta}$ that minimize the number of transmissions subject to memory constraints on nodes.

A. Buffer Assignment

Through an illustrative example (Fig. 3), we show our buffer allocation technique in the rest of this section.

The amount of data that can be stored within buffers is constrained by the available memory of the nodes. Different processing units, including reading (raw sampled sensor data), segmentation, feature extraction and classification produce data blocks of different sizes.

***Definition** 3:* The amount of data produced by unit u_i^μ per action is called *data unit* and denoted by b_i^μ. In the example shown in Fig. 3, reading, segmentation, feature extraction and classification generate data blocks of size 200, 2, 10 and 1 byte respectively.

Our communication model maintains two types of buffers for each sensor node as follows. A buffer of size b_i^μ associated with processing unit u_i^μ is called *intra-node buffer* in order to enable in-node processing. The data stored in this buffer will be consumed by the next processing unit within the same node. In the above example, the total amount of intra-bode buffers, labeled by W, maintained for nodes s_1 and s_2 is $W_1 = W_2 = 200 + 10 + 2 + 1 = 213$. Furthermore, a buffer allocated to a processing unit u_i^μ due to inter-node dependencies is called *inter-node buffer*. These buffers provide data for consequent dependent units.

Inter-node buffers are sized according to the number of actions each link would store. An inter-node link with $x_{ij}^{\mu\eta}$ number of actions allocates a buffer to the source unit and another buffer to the destination unit. When a source unit has more than one outgoing edge, only one buffer is enough to store data for both links. The size of such buffer is determined based on maximum amount of data required to be transmitted among the links. In Fig. 3, the processing unit u_1^S has two outgoing edges. The inter-node buffer (labeled by *B*) is then sized according to the values of x_{12}^{SS} and x_{12}^{SF}. However, only a single buffer with the maximum required size would be sufficient to accommodate the data generated by this unit. That is, a buffer of size $\max(2x_{12}^{SS}, 2x_{12}^{SF})$ is allocated where the coefficients are determined as the absolute number of data units that is required per processing block per action

(e.g. segmentation block generates two data units for each action). Destination units, however, require $2x_{12}^{SS}$ and $2x_{12}^{SF}$. Moreover, an inter-node dependency prevents the destination unit from performing any processing until it receives data from the source. That is, for link $e_{ij}^{\mu\eta}$, the data produced by the predecessor of the destination unit ($b_j^{\eta-1}$) must be buffered as well. In Fig. 3, link e_{12}^{SS} enforces a buffer of size $200x_{12}^{SS}$ on the incoming link e_{11}^{RS} because sensor reading produces 200 bytes. For the same reason, the $2x_{21}^{FF}$ bytes buffer on e_{11}^{SF}, and $10x_{12}^{CC}$ bytes buffer on e_{22}^{FC} are allocated and sized.

We make here certain assumptions about the frequency of occurrence of the actions. We assume that the number of movements occurring in a given time period T follows a Poisson distribution [16]. The probability of observing x number of actions in a given time interval is represented by

$$p(x) = \frac{\lambda^x e^{-\lambda}}{x!} \tag{4}$$

where λ is the expected number of movements occurred in the given interval. With a confidence level of $1-\alpha$, the Poisson model provides an upper bound k on the number of actions occurred during the interval:

$$p(x \leq k) = \sum_{x=0}^{k} p(x) = 1 - \alpha \tag{5}$$

Definition 4: The upper bound on the number of actions occurred during a given time interval is known as *action rate*, k, which is obtained according to Poisson distribution model.

B. Problem Formulation

Given the dependency graph G as described earlier, let $d_{ij}^{\mu\eta}$ be a binary that represents existence of inter-node links and is given by

$$d_{ij}^{\mu\eta} = \begin{cases} 1, & \text{if } s_i \text{ and } s_j \text{ are dependent through } u_i^{\mu} \text{ and } u_j^{\eta} \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

The number of actions A occurred at action rate k during time period T is given as $A = k \times T$, and the number of transmissions can be calculated by

$$Z = \sum_{i \neq j} \sum_{\mu,\eta} A \frac{d_{ij}^{\mu\eta}}{x_{ij}^{\mu\eta}} \tag{7}$$

The total size of the intra-node buffers W_i for node s_i is given by

$$W_i = \sum_{\mu} b_i^{\mu} \tag{8}$$

and the total size of the inter-node buffers for node s_i is determined by

$$B_i = \sum_{\mu} (\max_{j,\eta}(d_{ij}^{\mu\eta} b_i^{\mu} x_{ij}^{\mu\eta}) + \max_{j,\eta}(d_{ji}^{\eta\mu} b_j^{\eta} x_{ji}^{\eta\mu}) + \max_{j,\eta}(d_{ji}^{\eta\mu} b_j^{\mu-1} x_{ji}^{\eta\mu})) \tag{9}$$

Let M_i be the size of the memory on node s_i. The problem of minimizing the number of transmissions can be formulated as a convex optimization problem as follows:

$$\min \quad Z \tag{10}$$

subject to:

$$W_i + B_i \leq M_i \qquad \forall i \in \{1, ..., n\} \tag{11}$$

$$x_{ij}^{\mu\eta} \in \mathbb{Z}^+ \qquad \forall i, j, \mu, \eta \tag{12}$$

C. Problem Complexity

Both A and d_{ij} in (7) are known apriori. Thus, the objective function in (10) is to minimize $\sum_{j=1}^{m} \frac{1}{x_j}$. Furthermore, the non-linear constraints in (11) can be written as linear constraints by expanding the *max* functions in (9) over all possible values of j and η. With these modifications, the optimization problem in (10)-(12), can be represented as the following problem **P**:

$$\min \quad w(x) = \sum_{j=1}^{m} \frac{1}{x_j} \tag{13}$$

subject to set of feasible solusions F:

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{im}x_m + W_i \leq M_i \qquad \forall i \in \{1, \ldots, n\} \tag{14}$$

$$x_j \in Z^+ \qquad \forall j \in \{1, \ldots, m\} \tag{15}$$

In the above formulation, each x_j is associated with one of the existing inter-node links in the graph. The coefficients a_{ij} in (14) are calculated according to the transformation of non-linear constraints in (9) into linear equations by expanding every *max* function over all values taken by the function. Therefore, the set of feasible solutions can be represented by vectors of the form $\mathbf{x} = \{x_1, x_2, \ldots, x_m\}$. The new formulation in (13)-(15) will simplify our subsequent discussions.

The objective function $w(x)$ is a summation over several convex functions, turning the objective function into a convex separable function. To the best of our knowledge, no polynomial solution exists that solves this problem in its full generality. However, polynomial solutions exist for several special cases. In particular, authors in [17] present a greedy algorithm when the set of feasible solutions form a jump system. In another study, Hochbaum et al. [18] show that if the integer linear version of an integer convex separable minimization problem over linear constraints is polynomial, it can be solved in polynomial time in the number of variables m, the number of constraints, and the absolute value of the largest subdeterminant of the constraints matrix.

Since the IP problem is hard to solve in general, we propose two methods for approximating the solution. First, the integrality condition (15) can be relaxed as in (16) to solve the problem using common convex programming tools.

$$x_j > 0 \qquad \forall j \in \{1, \ldots, m\} \tag{16}$$

The solution obtained due to integrality relaxation will not carry the optimality condition, but we find a lower bound on the size of memory for which the result is optimal.

Theorem 1: For each sensor node s_i, the convex optimization problem **P** with integer relaxed constraints finds optimal solutions for memories of size

$$M_i - \sum_{j=1}^{m} (1 - \varepsilon) a_{ij} \tag{17}$$

Algorithm 1 Greedy solution for buffer allocation problem

Require: Optimization problem **P**:(w,F) and initial solution $\mathbf{x}^0 \in F$
Ensure: Solution **x** for **P**
  $\mathbf{x} \leftarrow \mathbf{x}^0$
  while $\exists St(\mathbf{x}+s)$ s.t. $\mathbf{x}+s \in F$ and $w(\mathbf{x}+s) < w(\mathbf{x})$ **do**
    $\hat{s} \leftarrow \arg\min_s w(\mathbf{x}+s)$
    $\mathbf{x} \leftarrow \mathbf{x}+\hat{s}$
  end while
  return **x**

Proof: Relaxation of a variable x_j, will round the variable down to closest integer. Thus, the size of optimal buffer associated with x_j would increase by factor $1-\varepsilon$ of the coefficients a_{ij} represented in (14). Therefore, memory usage on each node is optimized as shown in (17). ■

In investigating polynomial time solutions for the buffer assignment, we further develop a greedy algorithm which has less computational complexity than the IP technique.

V. Greedy Algorithm

In this section, we present a greedy solution for solving the buffer assignment problem. Our algorithm starts with an initial feasible solution and reiterates an augmentation process which moves greedily toward the final solution. Since the algorithm is motivated by several basic concepts mostly used in combinatorial optimization over jump systems, we first define these basic terms as follows.

***Definition** 5:* For any two lattice points [19] **x** and **y** in Z^m, the box spanned by **x** and **y** is denoted by [**x**,**y**] and defined as the set

$$\{\mathbf{z} \in \mathrm{Z}^m | \min(x_j, y_j) \le z_j \le \max(x_j, y_j)\} \quad \forall j \in \{1, ..., m\} \tag{18}$$

The spanned box [**x**,**y**] contains all the points **z** such that

$$d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{y}) = d(\mathbf{x}, \mathbf{y}) \tag{19}$$

where $d(\mathbf{x}, \mathbf{y})$ denotes the Manhattan distance between two vectors **x** and **y**, and is given by

$$d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|_1 = \sum_{j=1}^{m} |x_j - y_j| \tag{20}$$

***Definition** 6:* For any two points **x** and **y** in Z^m, a *step* s from **x** to **y** is a point $\mathbf{x}' \in \mathrm{Z}^m$ such that $\mathbf{x}' \in [\mathbf{x}, \mathbf{y}]$ and $d(\mathbf{x}, \mathbf{x}') = 1$.

The definition of step implies that in any step from **x** to **y**, only one of the components x_j should change. We denote this component by s (i.e. $\mathbf{x}' = \mathbf{x} + s$) and refer to such step St(**x**+s) as used in [17].

Our greedy approach is shown in Algorithm 1. It takes the convex optimization problem **P:** (w,F) and an initial feasible solution $\mathbf{x}^0$ as input and generates a final solution **x**. We will compare the results of this approach with the results obtained from integer programming in the experimental results section. The algorithm iterates through subsequent steps as follows. At each time, it checks all components x_j within vector **x** and finds the step St(**x**+s) that minimizes the objective function w. It then updates the current vector **x** and chooses the next step by setting $\mathbf{x} \leftarrow \mathbf{x} + \hat{s}$. The algorithm terminates when no other feasible solution exists that improves the objective function. This happens when either all following steps would violate the constraints F or every feasible step increases the value of $w(x)$.

***Lemma** 2:* Any step s from a current feasible solution $\mathbf{x}^{curr}$ to the final solution $\mathbf{x}^{final}$, updates the objective function w only by increasing one variable x_j.

Proof: Assume step s takes the algorithm for $\mathbf{x}^{curr}$ to a next step $\mathbf{x}^{next}$ updating the objective function $w(\mathbf{x}^{curr})$ to $w(\mathbf{x}^{next})$. This means $w(\mathbf{x}^{next}) < w(\mathbf{x}^{curr})$. By definition of the step, $d(\mathbf{x}^{curr}, \mathbf{x}^{next}) = 1$. To keep the distance between the two vectors 1, one component x_j should be either increased or decreased by 1. However, decreasing x_j would result in increasing the value of the objective function ($w(\mathbf{x}^{next}) > w(\mathbf{x}^{curr})$) which contradicts greedily minimizing the objective function w. Therefore, any step s increments one component of the vector **x**. ■

***Definition** 7:* Within each step s of the greedy algorithm, an *augmented link* is defined as the inter-node edge e_j whose corresponding variable x_j is incremented.

A. Static Buffer Allocation

The greedy algorithm can be used offline to determine initial buffers associated with different links. This is required at the system startup where dependencies are fixed and defined by the application. In addition to the objective function w and feasible solutions F defined by (13)-(15), the algorithm needs an initial feasible solution $\mathbf{x}^0$ as input. An obvious value for $\mathbf{x}^0$ is (1, 1, ..., 1) which is obtained by initializing every variable x_j to the smallest possible value introduced by (15). As a result, we initialize all x_j to 1 at the beginning of the program. At each iteration, the algorithm will increment the variable (see Lemma 2) that maintains locally optimum solution. Each link augmentation allocates some buffer to both processing blocks connected by the link as discussed in section IV-A.

***Lemma** 3:* Unless memory constraints in (14) are violated, in each round of the greedy algorithm all dependency links are augmented.

Proof: By induction on value of variables x_j. At the beginning of the algorithm, x_j=1 for all the links x_1, x_2, ... x_m. Let w_1 be the value of the objective function by augmenting all the links, through one round of the algorithm (e.g. m steps). Further, let w_1' be the objective function by augmenting all the links, but x_l which remains unchanged (x_l = 1) and x_p which is increased by 2, through m iterations. Therefore,

$$\begin{aligned} w_1 &= \sum_{j=1}^{m} \frac{1}{x_j+1} = \frac{m}{2} \\ w_1' &= \frac{1}{x_l} + \frac{1}{x_p} + \sum_{j=1, j\neq l,p}^{m} \frac{1}{x_j+1} = \frac{m}{2} + \frac{5}{6} \end{aligned} \tag{21}$$

Obviously, $w_1 < w_1'$ which means the algorithm will evenly augment the links. Now assume, in the kth round of the algorithm, all variables x_1=x_2=...=x_m=k increased by 1. We need to prove that in the $(k+1)$th round all the links will be augmented. Let w_{k+1} be the result of augmenting all the links (x_j=$k+1$), and w_{k+1}' the value of the objective function by

augmenting all the links except x_l which remains $k+1$ and x_p which is augmented twice (x_p=$k+3$). Thus,

$$w_{k+1} = \sum_{j=1}^{m} \frac{1}{x_j + 1} = \frac{m}{k+2} \tag{22}$$

$$\begin{aligned} w'_{k+1} &= \frac{1}{x_l} + \frac{1}{x_p} + \sum_{j=1, j\neq l,p}^{m} \frac{1}{x_j + 1} \\ &= \frac{m}{k+2} + \frac{2}{(k+1)(k+2)(k+3)} \end{aligned} \tag{23}$$

Since $w_{k+1} < w'_{k+1}$ the algorithm will augment all inter-node links evenly in any round of the algorithm. ■

***Theorem** 4:* The greedy algorithm for a given buffer allocation problem **P** as in (13)-(15) is polynomial in the number of dependency links *m* and the size of memory M.

Proof: Without loss of generality, assume all the sensor nodes have equal memory sizes M and within-node buffers of size W. Also, assume that the algorithm starts with an initial solution $\mathbf{x}^0$ which holds inequality constraints as described previously. The algorithm takes steps that minimize the objective function in (13). This convex separable function forces the algorithm to choose the next step such that variables x_j get close to each other. Base on Lemma 3, the greedy approach would make all x_j equal unless incrementing a variable violates some constraints. This procedure can be interpreted as follows. The algorithm takes m steps to augment each of the m inter-node links. For each node s_i, the amount of memory assigned by this round of the algorithm is given by

$$\theta_i = \sum_{j=1}^{m} a_{ij} \tag{24}$$

where a_{ij} are given by the constraints in (14). The remaining memory (M-W-θ_i) will be further assigned to different buffers in following steps. When increasing a variable x_j violates constraints, the algorithm ignores that variable from further buffer allocation. The number of such rounds that eliminate variables from future augmentation differs from one variable to another. However, in worst case, a link may require $\frac{M-W}{\theta_i}$ rounds before running out of memory. Because each round of the algorithm takes at most m steps, the number of iterations is bounded by $m \times \frac{M-W}{\theta_i}$. Assuming fixed W and θ_i for a given problem, the total number of steps St(**x**+s) required for the algorithm to converge has a complexity of $O(m \times M)$ ■

B. Dynamic Buffer Allocation

The goal of dynamic buffer allocation is to reassign buffers based on changes in dependency graph. At each point in time, one or more inter-node links may be removed from the graph and several edges can be added. Since the system is expected to operate in real-time, it is extremely important to quickly decide what the size of each buffer should be. In this section, we introduce a simple and effective approach for dynamic assignment of the buffers based on our greedy algorithm. Deletion of an inter-node link makes some memory available on the nodes. The new memory space can be used to assign buffers to the existing or newly inserted links. However, the optimality of the solution is not guaranteed if only memory of the deleted links is taken into consideration when reassigning buffers. To overcome this problem, we explore certain properties of our greedy algorithm which allow us to drastically reduce the number of steps for real-time buffer allocation. The key idea is that, by means of information obtained from the static case, the algorithm can restart with a significantly large initial feasible solution and yet achieve the same solution as the static buffer allocation approach.

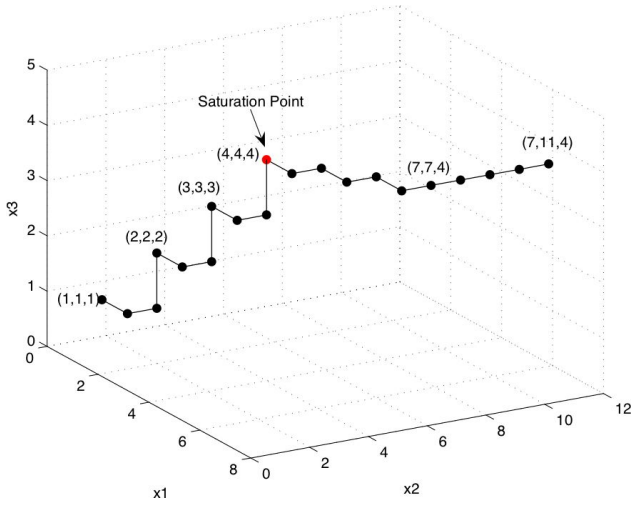

Fig. 4. **Illustration of saturation point**

***Definition** 8:* Given probelm **P** with objective function w as in (13) and feasible solutions F in (14)-(15), and initial solution $\mathbf{x}^0$ = (1, 1, …, 1), a *saturation point* is a point $\mathbf{x}^{sat}$ within spanned box [$\mathbf{x}^0$,**x**] (**x** $\in F$) such that all components of $\mathbf{x}^{sat}$ have the same value v^{sat}, and v^{sat} is maximized.

$$\mathbf{x}^{sat} = (v^{sat}, \ldots, v^{sat}) \tag{25}$$

$$v^{sat} = \arg\max_{v}(v_1, \ldots, v_m) \in F \qquad \forall j \;\; v_j = v \tag{26}$$

When the greedy algorithm iterates through steps, different links will be saturated one after another. Once a link is saturated, it is ignored in the subsequent iterations. The saturation point, in fact, refers to the first link that is eliminated from further augmentation. Fig. 4 shows evolution of the algorithm for three variables x_1, x_2, and x_3. The algorithm starts with the initial solution (1,1,1) and reiterates by evenly incrementing the variables. The point (4,4,4) is the saturation point because all variables have the same value and one of the links (x_3) is saturated. This link can not be further augmented in following steps due to memory constraints. The algorithm continues by augmenting other links than x_3 until x_1 becomes saturated at (7,7,4). However, it keeps increasing x_2 until the algorithm converges at (7,11,4).

***Lemma** 5:* Given final solution **x**=(v_1, …, v_m) obtained from the greedy algorithm, the saturation point $\mathbf{x}^{sat}$ = (v^{sat}, …, v^{sat}) is calculated by

$$v^{sat} = \min(v_1, \ldots, v_m) \tag{27}$$

Proof: The proof follows from definition of the saturation point. Saturation point is identified by the first link that is saturated. Since such link will not be augmented in future, its value remains fixed throughout subsequent steps. Therefore,

at the end of the algorithm, the value of such link is the same as its last augmentation, but other links have been augmented in subsequent iterations (see proof for Theorem 4). ■

1) Link Deletion: Removing an existing link will create free space for other links to be augmented. On deletion of a link, we run our greedy algorithm with the initial feasible solution given by the saturated point. This guarantees the same final solution as the static algorithm and reduces the number of steps required for the algorithm to converge.

***Lemma** 6:* Given a buffer allocation problem **P**, any point in the box spanned by two feasible solutions **x** and **x′** remains a feasible solution if any memory size M_i increases.

Proof: Let **P** be the problem which allocates buffers to sensor nodes $s_1, \ldots s_n$ with memories of size $M_1, \ldots, M_n$. The set of feasible solutions F for problem **P** is defined by the box spanned by the smallest initial solution $\mathbf{x}^0$ and the optimal solution $\mathbf{x}^{opt}$. Increasing M_i by δ will expand this box according to the constraints in (14). Therefore, the new expanded box contains all previous feasible solutions. In particular, it contains all the points within the box [**x**,**x′**]. ■

***Theorem** 7:* By removing a link from dependency graph, the greedy algorithm with initial solution $\mathbf{x}^{sat}$ produces the same result as that with the initial solution $\mathbf{x}^0$ used for static approach.

Proof: Let v^{sat} refer to the saturation point for the original problem with m links. Without loss of generality, assume that link e_m has been removed from dependency graph. The problem turns into finding final solution for **x**=(x_1, …, x_{m-1}). Let w and w' be the values of the objective function using initial solutions $\mathbf{x}^0$=(1, …, 1) and $\mathbf{x}^{sat}$=(v^{sat}, …, v^{sat}) respectively. Because link deletion makes some memory available, any point within the box [$\mathbf{x}^0$,$\mathbf{x}^{sat}$] is a feasible solution (see Lemma 6). Thus, $\mathbf{x}^{sat}$ could be an initial point for the greedy algorithm. Based on Lemma 3, this point is on the path from $\mathbf{x}^0$ to the final solution. Therefore, both static and dynamic approaches will take the same steps after reaching $\mathbf{x}^{sat}$, and output the same solutions. ■

2) Link Insertion: When a new link is added to the network, we still use the saturation point to restart the algorithm. However, this point would not necessarily be a feasible solution for the new problem with extra link. The reason for this is that the new variable added to the constraints in (14) may shrink the feasible solution space. Yet, we are interested in finding an initial point whose all components have the same value. This point would necessarily lie within the box [$\mathbf{x}^0$,$\mathbf{x}^{sat}$]. To find this initial point, we perform a quick binary search. This gives a feasible point in [$\mathbf{x}^0$,$\mathbf{x}^{sat}$] which has largest components. Once this point is found, we feed it to our greedy algorithm. The algorithm will then go through different steps by evenly augmenting existing links. When incrementing a variable results in violating memory constraints, the algorithm skips that link from further augmentation and iterates through the rest of edges in the dependency graph as accomplished for the static case.

VI. Experimental Verification

In this section, we show the effectiveness of our buffer assignment techniques through several networks with varying inter-node dependencies.

TABLE I
NUMBER OF TRANSMISSIONS FOR EXPERIMENT 1 FOR 24 HOURS OF SYSTEM OPERATION

Conf.	#Packets (W/o buffer)	ILP		Greedy	
		#Packets	%Saving	#Packets	%Saving
a	2880	52	98.19%	89	96.91%
b	2880	2	99.93%	2	99.93%
c	2880	41	98.58%	88	96.94%
d	2880	36	98.75%	58	97.99%
e	2880	73	97.47%	128	95.56%
f	2880	9	99.69%	9	99.69%
Avg	2880	35.50	98.77%	62.33	97.84%

A. Setup

We conduct our experiments on several networks, each composed of homogenous sensor nodes. The nodes are TelosB as described in section III-A and have memories of size 10 KB. The results are discussed for two different networks with respect to typical dependencies required in BSN applications. Here x_j denotes the number of actions for which data is buffered before transmission over the given link.

In our analysis, we assume that the movements occur at a rate of 1 action/min. and each movement spans 200 samples. These assumptions ensure a high confidence level (95%) with the Poisson model. In fact, processing blocks will have sufficient computational throughput to process incoming actions. We further assume that the size of intra-node buffer is 2 bytes for segmentation, 10 bytes for feature extraction, and 1 byte for classification.

B. Experiment 1

For the first experiment, we arranged a network of two sensor nodes with emphasis on information dependencies for segmentation and classification as shown in Fig. 5(a). A brief description of this configuration is as follows. The second node is the master node for segmentation and classification. It performs segmentation based on its local knowledge and the information received from the node 1. In order to provide the first node with a more confident segmentation, node 2 transmits the results to node 1 for feature extraction. Furthermore, the classifier at node 2 is assumed to be dependent on the results of classification provided by node 1.

The network is initially constructed to satisfy the three information dependencies labeled as x_1, x_2 and x_3. Fig. 5(b) shows the situation where x_1 and x_2 are removed from the network. The network model is then updated as shown in Fig. 5(c) due to addition of two new edges x_4 and x_5. The network changes afterwards as shown in Fig. 5(d), Fig. 5(e) and Fig. 5(f).

We solved the buffer allocation problem using both integer relaxation and greedy algorithm. The IP solver in [20] was used to specify and solve the convex optimization problem. To evaluate the effectiveness of our combinatorial algorithm, we further developed the greedy algorithm in MATALB. In the absence of buffers, each node is required to transmit its local data to the other node on occurrence of every action. This results in 1440 transmissions per node during twenty-four hours of system operation. Table I shows difference between the relaxed ILP and greedy buffer allocations for each of the six configurations in Fig. 5. When network model changes

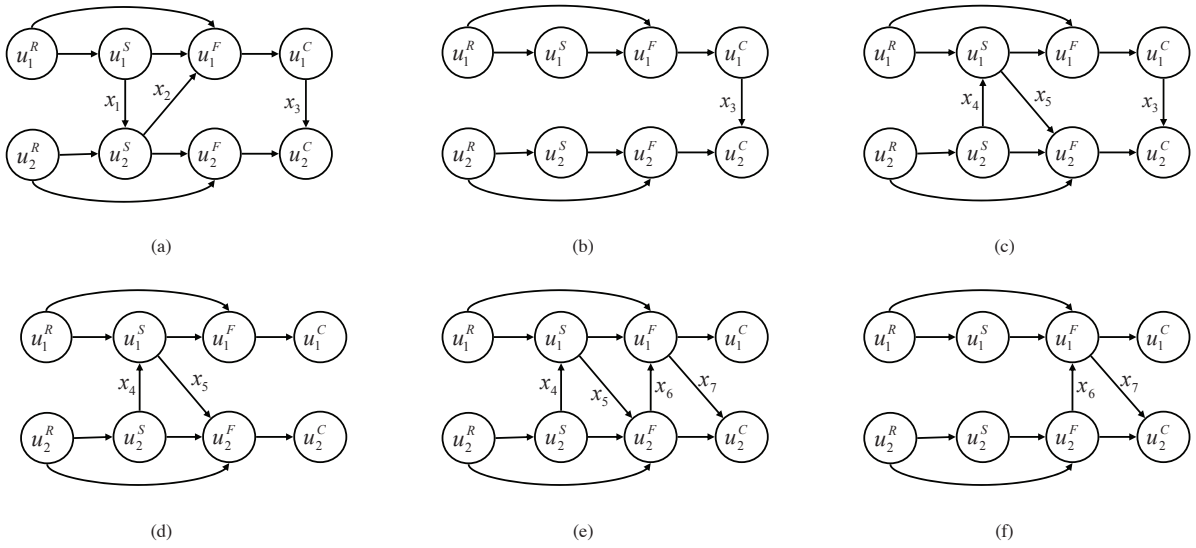

Fig. 5. Evolution of dependency graph for a body sensor network with two sensor nodes (used for experiment 1)

TABLE II
TIME COMPLEXITY COMPARISON FOR EXPERIMENT 1

Conf.	ILP Time (sec.)	Greedy Static			Greedy Dynamic			
		Time (μsec)	#Steps	Speedup	Time (μsec)	#Steps	Speedup(ILP)	Speedup(Static)
a	1.453	188	144	7728	-	-	-	-
b	1.234	3172	911	389	2750	863	448	1.15
c	1.625	206	145	7888	15.6	1	104166	13
d	1.518	196	98	7744	27	2	56222	7.2
e	2.030	294	176	6904	20	4	101500	14.7
f	1.478	1978	668	747	1740	1160	849	1.1

TABLE III
NUMBER OF TRANSMISSIONS FOR EXPERIMENT 2 FOR 24 HOURS OF SYSTEM OPERATION

Conf.	#Packets (W/o buffer)	ILP		Greedy	
		#Packets	%Saving	#Packets	%Saving
a	4320	81	98.13%	194	86.5%
b	4320	59	98.63%	58	95.9%
c	4320	95	97.80%	128	91.1%
d	4320	6	99.86%	6	99.5%
e	4320	117	97.29%	191	86.7%
f	4320	71	98.36%	116	91.9%
Avg	4320	71.50	98.34%	115.5	97.33%

from one configuration to another (e.g. from (a) to (b)), both static and dynamic algorithms achieve the same results. For this particular experiment, the ILP and greedy reduced the number of transmissions by an average factor of 81 and 46 respectively. Since the sensor nodes are incapable of running the ILP, we run our algorithms on a PC to provide insights into differences in terms of running time between the greedy schemes and the ILP. For this purpose, the convex solver and the developed tools were used in MATLAB running on a laptop with a 1.6GHz Intel Core Duo processor. The results are shown in Table II which verifies the effectiveness of our algorithm in terms of convergence time. Number of steps in this table corresponds to Definition 6. For each of the (b) thru (f) networks, the last two columns in Table II shows the speedup of the dynamic algorithm compared with that of the ILP and static buffer allocation schemes.

C. Experiment 2

The second network that we used to evaluate our techniques is illustrated in Fig. 6. The initial configuration shown in Fig. 6(a) has six inter-node dependency links (labeled as x_1 to x_6) for segmentation, feature extraction, and classification. Fig. 6(b) thru 6(c) illustrate dynamic changes in the network due to potential requirements of the application. First, the network loses its all inter-node dependencies for feature extraction and classification (x_2, x_3, x_5, and x_6), but not for segmentation. Network configuration is then updated as in Fig. 6(c) to enable classification at the second node. In this configuration, the second classifier is assumed to combine features from all the sensors for high accurate movement classification. In Fig. 6(d), the network has only dependencies for classification at node 2, as a result of deleting links x_1 and x_4 from previous network. We keep updating the model as shown in Fig. 6(e) by adding four new dependencies (x_9, x_{10}, x_{11}, and x_{12}) which makes the second node master for segmentation. Node 2 receives segmentation data from the two other nodes and provides them with final annotations. Fig. 6(f) shows situation where node 2 is not a classifier combiner as it was in (c), (d) and (e). This is done by deleting x_7 and x_8 from the network. However, node 2 is still master for segmentation.

Note that when no buffering scheme is employed, a transmission is initiated after occurrence of each movement. This yields a large number of transmissions as shown in Table III. This table also shows the number of transmissions reported by both ILP and greedy solutions. For this experiment, the number of transmissions is reduced by a factor of 60 and 37

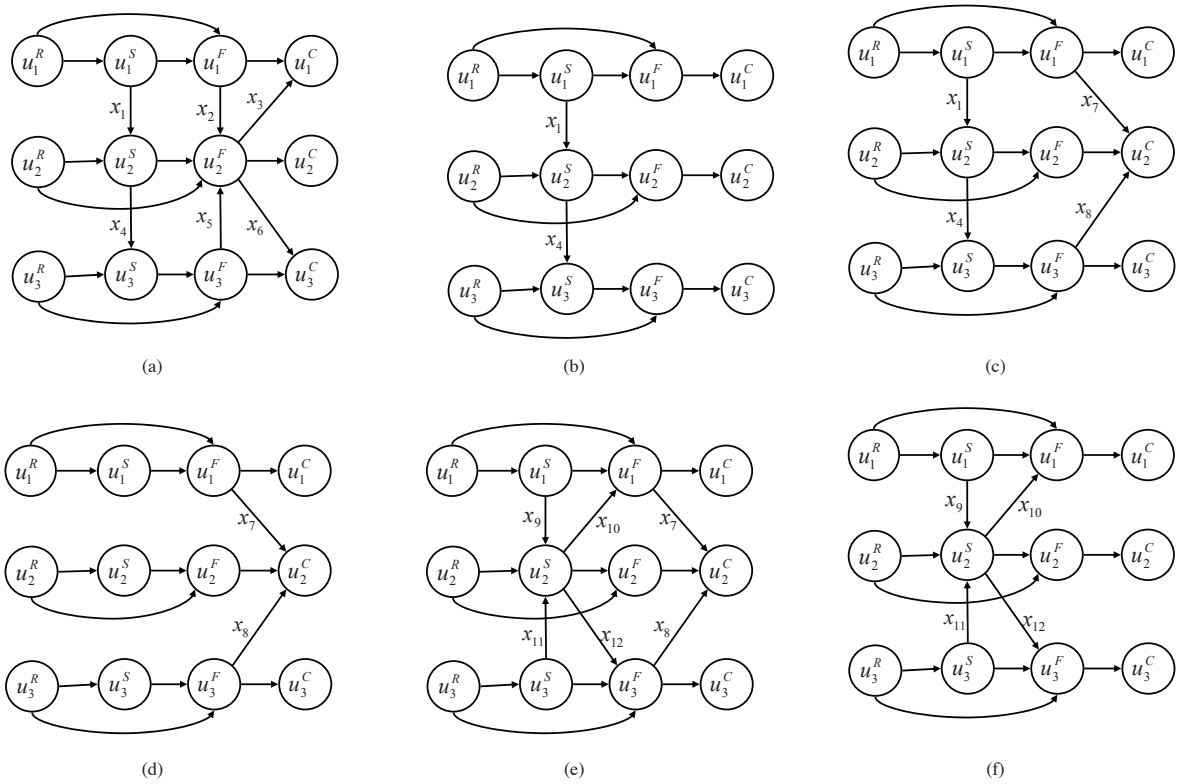

Fig. 6. Example of dependency graph for three sensor nodes with dynamic reconstruction of inter-node links (used for experiment 2)

TABLE IV
TIME COMPLEXITY COMPARISON FOR EXPERIMENT 2

Conf.	ILP Time	Greedy Static			Greedy Dynamic			
	(sec.)	Time (μsec)	#Steps	Speedup	Time (μsec)	#Steps	Speedup(ILP)	Speedup(Static)
a	2.532	6266	261	404	-	-	-	-
b	1.484	2343	98	633	297	12	4996	7.8
c	2.078	4031	176	515	109	4	19064	36
d	1.612	8563	1002	188	8175	914	1763	1.09
e	2.555	3520	266	725	150	2	17033	23
f	2.152	2872	196	749	109	20	19743	26

using the ILP-based and greedy respectively. An interesting observation is that for Fig. 6(b) the ILP solver outputs a larger number of packets than the greedy algorithm. This is, in fact, due to integer relaxation of the variables x_j when using convex solver (see Theorem. 1).

In Table IV, we compare timing complexity of different solutions including ILP and static and dynamic greedy. As demonstrated by our results, the greedy algorithm can drastically reduce cpu-time while maintaining higher savings in terms of packet transmissions. Moreover, real-time allocation of buffers using our dynamic greedy serves considerable time reduction compared to both ILP and static greedy. This verifies importance of employing saturation point as initial solution of the greedy algorithm for dynamic reconfiguration.

VII. DISCUSSION AND FUTURE WORK

Our model provides a unique way for energy conservation, context-aware and collaborative communication systems that utilizes specific properties of signal processing applications in BSNs. The proposed technique has several advantages in addition to the main objective of transmission reduction. It can simplify the communication, reduce the overhead of communication coordination, and enhance the system lifetime.

Although the existence of real-time deadlines is not investigated in this study, our model can be suitably extended further to incorporate such need. Immediate reactions can be initiated when certain actions (e.g. falling) are observed. For each such action, one master node can detect the movement locally and initiate the communication.

The choice of the number of packets as a performance metric makes our evaluation independent of the communication protocol. Moreover, larger packet sizes presented by our model can potentially enhance communication system by reducing energy consumption and supporting self-configuration [21].

In future, we will investigate accommodating deadlines in our communication model. Furthermore, we plan on refinement of our greedy algorithm for faster execution by incorporating more intelligent approaches that use larger step sizes.

VIII. Conclusion

In this paper, we outlined a communication model in order to reduce the number of transmissions over the BSN networks. We made use of an efficient buffer assignment technique in distributed light-weight sensory systems with limited storage. Our technique maps the communication minimization model into a convex optimization problem and provides an integer programming as well as a greedy algorithm to calculate buffer sizes associated with each processing module. We further developed a dynamic resource allocation technique based on our greedy approach which reduces time required for buffer assignment in real-time. Our model can be used to realize different BSN applications with relaxed processing deadlines.

References

[1] V. Raghunathan, C. Schurgers, S. Park, and M. Srivastava, "Energy-aware wireless microsensor networks," *IEEE Signal Processing Mag.*, vol. 19, no. 2, pp. 40–50, 2002.

[2] J. Luprano, J. Sola, S. Dasen, J. M. Koller, and O. Chetelat, "Combination of body sensor networks and on-body signal processing algorithms: the practical case of myheart project," in *BSN '06: Proceedings of the International Workshop on Wearable and Implantable Body Sensor Networks*, 2006, pp. 76–79.

[3] J. M. Kahn, R. H. Katz, and K. S. J. Pister, "Next century challenges: mobile networking for "smart dust"," in *MobiCom '99: Proc. 5th annual ACM/IEEE international conference on Mobile computing and networking*, 1999, pp. 271–278.

[4] R.-S. Liu, K.-W. Fan, and P. Sinha, "Locally scheduled packet bursting for data collection in wireless sensor networks," *Ad Hoc Networks*, vol. 7, no. 5, pp. 904 – 917, 2009.

[5] C. Chigan and V. Oberoi, "Providing qos in ubiquitous telemedicine networks," in *PERCOMW '06: Proc. 4th annual IEEE international conference on Pervasive Computing and Communications Workshops*, 2006, p. 496.

[6] H. Ghasemzadeh, E. Guenterberg, K. Gilani, and R. Jafari, "Action coverage formulation for power optimization in body sensor networks," in *Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific*, 2008, pp. 446–451.

[7] Y. Liu, B. Veeravalli, and S. Viswanathan, "Critical-path based low-energy scheduling algorithms for body area network systems," Aug. 2007, pp. 301–308.

[8] H. Ghasemzadeh, J. Barnes, E. Guenterberg, and R. Jafari, "A Phonological Expression for Physical Movement Monitoring in Body Sensor Networks," in *Mobile Ad Hoc and Sensor Systems, 2008. MASS 2008. 5th IEEE International Conference on*, 2008, pp. 58–68.

[9] H. Ghasemzadeh, V. Loseu, and R. Jafari, "Structural action recognition in body sensor networks: Distributed classification based on string matching," *IEEE Trans. Inf. Technol. Biomed.*, 2009.

[10] A. Chandrakasan, R. Amirtharajah, S. Cho, J. Goodman, G. Konduri, J. Kulik, W. Rabiner, and A. Wang, "Design considerations for distributed microsensor systems," 1999, pp. 279–286.

[11] H. T. Kung and K. Chang, "Receiver-oriented adaptive buffer allocation in credit-based flow control for atm networks," in *INFOCOM '95: Proc. Fourteenth Annual Joint Conference of the IEEE Computer and Communication Societies (Vol. 1)-Volume*, 1995, p. 239.

[12] J. Hu, U. Y. Ogras, and R. Marculescu, "System-level buffer allocation for application-specific networks-on-chip router design," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 25, no. 12, pp. 2919–2933, Dec. 2006.

[13] J. Hahn and P. H. Chou, "Buffer optimization and dispatching scheme for embedded systems with behavioral transparency," in *EMSOFT '07: Proc. 7th ACM & IEEE international conference on Embedded software*. New York, NY, USA: ACM, 2007, pp. 94–103.

[14] A. Benveniste, P. Caspi, M. di Natale, C. Pinello, A. Sangiovanni-Vincentelli, and S. Tripakis, "Loosely time-triggered architectures based on communication-by-sampling," in *EMSOFT '07: Proc. 7th ACM & IEEE international conference on Embedded software*. New York, NY, USA: ACM, 2007, pp. 231–239.

[15] H. Ghasemzadeh, N. Jain, M. Sgroi, and R. Jafari, "Communication minimization for in-network processing in body sensor networks: A buffer assignment technique," in *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09.*, April 2009, pp. 358–363.

[16] A. Panangadan, M. Mataric, and G. Sukhatme, "Detecting anomalous human interactions using laser range-finders," *Intelligent Robots and Systems, 2004. (IROS). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3, pp. 2136–2141 vol.3, 2004.

[17] K. Ando, S. Fujishige, and T. Naitoh, "A greedy algorithm for minimizing a separable convex function over a finite jump system," *Journal of Operations Research*, vol. 38, no. 3, pp. 352–375, September 1995.

[18] D. S. Hochbaum and J. G. Shanthikumar, "Convex separable optimization is not much harder than linear optimization," *J. ACM*, vol. 37, no. 4, pp. 843–862, 1990.

[19] E. Krtzel, *Lattice Points*. Springer, 1988.

[20] M. Grant and S. Boyd, "Graph implementations for nonsmooth convex programs," Recent Advances in Learning and Control, (a tribute to M. Vidyasagar), V. Blondel, S. Boyd, and H. Kimura, editors, pages 95-110, Lecture Notes in Control and Information Sciences, Springer, 2008.

[21] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient mac protocol for wireless sensor networks," vol. 3, 2002, pp. 1567–1576 vol.3.

Hassan Ghasemzadeh received the B.Sc. degree in Computer Engineering from Sharif University of Technology, Tehran, Iran, and the M.Sc. degree in Computer Engineering from University of Tehran, Tehran, Iran in 1998 and 2001 respectively. He joined the University of Texas at Dallas in 2007 where he is pursuing his Ph.D. in Computer Engineering. His research interests lie in different aspects of embedded systems. He is currently working on collaborative signal processing, reconfigurable computing and algorithm design for medical embedded systems. He is a student member of the IEEE.

Vitali Loseu received the B.S. and M.S. degrees both in Computer Science from the University of Texas at Dallas in 2007 and 2008 respectively. He then proceeded to pursue his Ph.D. in Computer Engineering in the Embedded Systems and Signal Processing Lab (ESSP). His research interests lie in system optimization for reconfigurable computing. He is a student member of the IEEE.

Sarah Ostadabbas received her B.Sc. in both Electrical and Biomedical Engineering from Amirkabir University of Technology, Tehran, Iran, and the M.Sc. degree in Control Engineering from Sharif University of Technology, Tehran, Iran in 2005 and 2007 respectively. She is currently pursuing her PhD in Electrical Engineering at the University of Texas at Dallas. Her research interests are primarily embedded system and signal processing with an emphasis on medical/biological applications.

Roozbeh Jafari received his B.Sc. in Electrical Engineering from Sharif University of Technology in 2000. He received an M.S. in Electrical Engineering from SUNY at Buffalo, and an M.S. and a Ph.D. in Computer Science from UCLA in 2002, 2004 and 2006 respectively. He spent 2006-2007 in EECS department at UC Berkeley as a post-doctoral researcher. Dr. Jafari is currently an assistant professor in Electrical Engineering at the University of Texas at Dallas. His research is primarily in the area of networked embedded system design and reconfigurable computing with emphasis on medical/biological applications, their signal processing and algorithm design. He is the director of ESSP Lab.