A Greedy Buffer Allocation Algorithm for Power-aware Communication in Body Sensor Networks

Hassan Ghasemzadeh, Roozbeh Jafari
Department of Electrical Engineering
University of Texas at Dallas, Richardson, TX 75080-3021
{h.ghasemzadeh, rjafari}@utdallas.edu

ABSTRACT

Monitoring human movements using wireless sensory devices promises to revolutionize the delivery of healthcare services. In spite of their potentials for many application domains, power requirements and wearability have limited the commercialization of these systems. In this paper, we propose a novel approach for optimizing communication energy by reducing inter-node data transmissions. This is accomplished by introducing buffers that limit communication to short bursts, and therefore decrease power usage and simplify the communication. Our buffer allocation is a greedy algorithm that can operate both in a centralized and distributed architecture. We experimentally demonstrate the effectiveness of our power reduction techniques. Our results show that, compared with an unbuffered system, our system achieves more than 30% reduction in energy consumption.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special Purpose and Application-Based Systems—*Real-time and embedded systems*; J.3 [**Computer Applications**]: Life and Medical Science—*Health*; H.1.2 [**Information Systems**]: Models and Principles—*User/Machine Systems Human information processing; Human factors.*

General Terms

Design, Algorithms, Performance.

Keywords

Body Sensor Networks, Collaborative Signal Processing, Buffer Allocation, Communication, Energy.

1. INTRODUCTION

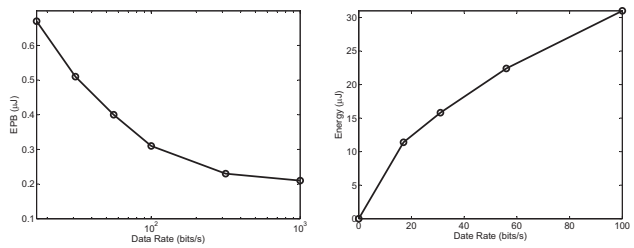
Body Sensor Networks (BSNs) are an emerging platform aiming to foster medical services by providing more options for remote patient monitoring, diagnosis, and care. This will give the patients more control over their treatment and increase the number who can be treated in-home. A BSN consists of several wireless sensor nodes placed at relevant locations on the patient. Each node consists of several sensors, an embedded processor, and radios for communication between nodes. Depending on the application of interest, certain processing is done to extract relevant information from data collected by individual nodes. This is accomplished through embedded signal processing modules that are implemented on the nodes, and function collaboratively with respect to the BSN application. Each of the processing components has overhead in memory, runtime, and communication. Because these processing blocks are developed for execution on light-weight embedded systems which are extremely constrained in terms of available resources, efficient design and implementation of embedded processing modules is crucial.

Efficient design and deployment of BSNs involves various challenges related to computation, communication, battery lifetime, memory, and wearability factors. Typically, applications utilize context-aware sensing and collaboration among nodes to increase accuracy using wireless. This collaboration incurs communication costs and can cause substantial reduction in battery life. Studies show that wireless communication is more expensive in terms of energy consumption than sensor data processing [1, 2]. Therefore, reducing communication is critical to achieving reasonable system lifetimes.

Transmitting data in bursts is an efficient power saving approach in wireless sensor network. It can reduce complexity of communication, lower energy consumption, and increase system throughput [3, 4]. An efficient burst communication methodology would minimize the number of transmissions. This requires exploration of memory architectures that allow for allocation of optimal size buffers on individual sensor nodes. Intuitively, data are accumulated in buffers and will be transmitted in bursts and at a higher rate. Several diagrams in Figure 1 support this claim. As shown in Figure 1(a), the amount of energy per bit (EPB) deceases as data rate grows. Figure 1(b) shows the total energy as a function of data rate for ZigBee. For visualization, the graph in Figure 1(b) is shown only for data rates 0-100 bps.

In this paper, we present a combinatorial algorithm for fast allocation of buffers in BSN platforms. We explore both memory and communication architectures to leverage our power optimization technique using buffer allocation. This work is motivated by medical monitoring applications that do not require immediate results. For example, in many

CODES+ISSS'10, October 24–29, 2010, Scottsdale, Arizona, USA.



(a) EPB (b) Energy

Figure 1: Energy per bit (EPB) and total energy cost versus data rate for ZigBee (based on energy model in [6])

applications physicians require periodic reports about their patients (e.g. how many times the wearer performed 'sit to stand' throughout the day), and therefore, real-time results are unnecessary [5]. Removing immediate data-processing deadlines permits an interesting optimization, where intermediate results are stored locally before being transmitted to other nodes.

2. RELATED WORK

The concept of buffers has been utilized in several domains in order to enhance communication. It has been traditionally used in ATM networks to improve system throughput by allowing communication parties to dynamically allocate their buffers to multiple nodes [7, 8]. In networks-on-chip (NoC), efficient distribution of buffers among input channels of on-chip routers can significantly increase overall performance of the system [9].

Buffers have been used in embedded software to minimize memory requirements of runtime software components. Authors in [10] introduce a buffer optimization technique based on synchronous data flow model. They represent lifetime of buffers using a 2-D tile formulation with different shapes corresponding to different process schedules. Minimum memory occupancy in time and space is obtained by exploring alternative schedules. In real-time distributed systems, buffers have been employed to avoid communication delay among synchronous processes and to preserve data semantics [11, 12]. To address implementation issues, authors in [13] discuss the problem of mapping synchronous models onto a network of nodes triggered by non synchronous clocks. While these techniques are successful in using buffers for semantic-preserving, they are not optimal in terms of memory usage. Authors in [14] generalize the above schemes to maintain buffers of minimum size when scheduling synchronous multi-task programs in a distributed fashion.

Our work explores a buffer allocation technique for BSN applications which require collaboration among embedded processing modules. Furthermore, the application does not have any immediate deadline for data transmissions. Authors in [15] introduced an ILP formulation of this problem aiming to maintain optimal size buffers across the network. Since computational complexity of the ILP limits its utilization in real-time scenarios, we present greedy-based approaches for buffer assignment in both centralized and distributed processing scenarios. Compared with the ILP solution, our greedy algorithms make considerable improvement in terms of computational complexity without sacrificing performance.

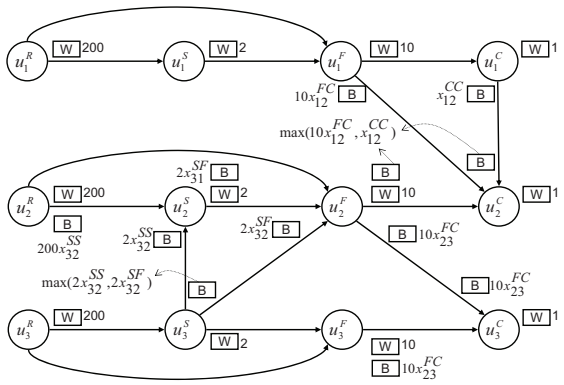


Figure 2: A task graph and buffer requirement over a network of three nodes.

3. PRELIMINARIES

Before introducing our algorithms for buffer allocation, we briefly present the architecture of our system followed by an application case study and typical collaborative signal processing in BSN applications.

3.1 Sensing Platform

A BSN consists of multiple body-worn sensor nodes, each equipped with a battery-powered microcontroller, a sensor board, and a radio for communication. For our experiments, we use TelosB motes which are commercially available from XBow®. Attached to each mote, we use a custom-designed sensor board consisting of a tri-axial accelerometer and a bi-axial gyroscope to capture human motions.

3.2 Collaborative Signal Processing

Our signal processing model involves several intra-node processing tasks including sampling/preprocessing, segmentation, feature extraction, and classification. The goal of this model is classification: determining what movement is being performed. In addition to intra-node data dependencies, the model has inter-node dependencies which enable collaboration among the nodes. We introduce this collaborative model using a *task graph* as shown in Figure 2.

Given n sensor nodes $\{s_1,\ldots,s_n\}$, inter-node dependencies are represented by a directed acyclic graph (DAG), G=(V,E). Each sensor node has a number of processing blocks, represented by vertices in the graph [15]. An example of such model for a network of three sensor nodes with five inter-node dependencies is shown in Figure 2. The four block types are denoted by R, S, F and C that correspond to sensor sampling/preprocessing, segmentation, feature extraction, and classification respectively. The edges from one sensor node to another represent data dependencies between different processing blocks. An example requiring inter-node communication is investigating movements of the upper body during walking. The leg node observes detailed walking data, whereas a node placed on the arm may not be able to identify the walking movement precisely. The segmented data from leg node can be used to perform the segmentation on the arm node.

4. PROBLEM STATEMENT

Buffers are introduced to minimize the number of data transmissions. The data is accumulated before transmission so that it can be sent at a high data rate over a short time interval. Figure 2 shows buffer allocation for a network configuration involving three nodes with five inter-node dependency links e_{12}^{FC}, e_{12}^{CC}, e_{32}^{SS}, e_{32}^{SF}, and e_{23}^{FC}. A variable $x_{ij}^{\mu\eta}$ is defined as the number of actions produced by the source module u_i^{μ} that are buffered prior to transmission to the destination unit u_j^{η}. The objective is to find $x_{ij}^{\mu\eta}$ that minimize the total number of transmissions subject to the memory constraints on each node.

Different processing units produce data blocks of different sizes. In the example of Figure 2, processing modules *reading*, *segmentation*, *feature extraction* and *classification* produce data blocks of sizes of 200, 2, 10, and 1 bytes for each action.

The system maintains two types of buffers depending on the links and adjacent vertices. Intra-node buffers, marked "W", contain information exactly for one action as required for in-node processing. The total number of these intra-node buffers for node s_1, s_2, and s_3 is $W_1 = W_2 = W_3 = 200 + 10 + 2 + 1 = 213$. The "$B$" buffers are inter-node buffers which are enforced by inter-node dependencies. The size of each such buffers is determined based on $x_{ij}^{\mu\eta}$ as described in the following.

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. For a processing unit with only one outgoing edge (e.g. u_2^F), one buffer is needed to store data produced by that unit. Also, another buffer of the same size is required on the consumer side to ensure receipt of the data. The size of such buffers is determined according to the data unit of the producer. Thus, for the link e_{23}^{FC}, two buffers of the size $2x_{23}^{FC}$ are allocated as shown in Figure 2. When a source 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 the maximum amount of data required by any of the links. In Figure 2, the processing unit u_3^S has two outgoing edges. The inter-node buffer (labeled by B) for this unit is sized according to the values of x_{32}^{SS} and x_{32}^{SF} and is equal to $\max(2x_{32}^{SS}, 2x_{32}^{SF})$, where the coefficients are determined as the absolute number of data units required per processing block per action (e.g. segmentation block generates two data units for each action). Destination units u_2^S and u_2^F, however, require buffers of size $2x_{32}^{SS}$ and $2x_{32}^{SF}$ respectively. Similarly, a destination unit with more that one incoming edge (e.g. u_2^C) requires a buffer sized according to the maximum amount of data enforced by the incoming links (e.g. $\max(10x_{12}^{FC}, x_{12}^{CC})$). However, since this unit receives data from separate producers, each incoming link needs different buffer at the producer size. For example, u_1^F and u_1^C maintain buffers of size $10x_{12}^{FC}$ and x_{12}^{CC} respectively.

While a processing block is waiting for data from one source, the required data from other sources cannot be processed due to dependency issues. Therefore, this data must be temporarily stored. For example, the link e_{32}^{SS} enforces a buffer of size $200x_{31}^{SS}$ on the incoming intra-node link e_{22}^{RS}. That is, the data produced by the predecessor of the destination unit ($b_j^{\eta-1}$) must be buffered as well.

Let A be the total number of actions that need to be detected during a fixed time interval (e.g. in 24 hours of system operation). The number of data transmissions for a link $x_{ij}^{\mu\eta}$ is equal to $\frac{A}{x_{ij}^{\mu\eta}}$. Therefore, the objective will be to minimize the total number of transmissions subject to the memory constraints on each node:

$$\text{Minimize} \quad \sum \frac{A}{x_{ij}^{\mu\eta}} \tag{1}$$

Subject to:

$$W_i + B_i \leq M_i \qquad \forall i \in \{1, ..., n\} \tag{2}$$

where for each node s_i, the variable W_i denotes the total amount of intra-node buffers, and M_i represents the total memory available at the node. Also M_i denotes the total available memory at node s_i. We note that the non-linear constraints can be written as linear constraints by expanding the *max* functions over all possible values of j and η. With these modifications, the above optimization problem can be represented as a convex optimization problem with linear constraints as will be discussed in the following section.

4.1 Problem Formulation

Given dependency graph G=(V,E) associated with a network of n sensor nodes s_1, s_2, ..., s_n that has m inter-node dependency links e_1, e_2, ..., e_m, the number of burst transmissions is inversely proportional to the buffer sizes allocated for all communication links. The problem of minimizing the number of transmissions can be formulated as a convex optimization problem:

$$\text{Minimize} \quad w(\mathbf{x}) = \sum_{j=1}^{m} \frac{1}{x_j} \tag{3}$$

subject to set of feasible solutions F:

$$\begin{aligned} a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{im}x_m + W_i \leq M_i \\ \forall i \in \{1, \ldots, n\} \end{aligned} \tag{4}$$

$$x_j \in Z^+ \quad \forall j \in \{1, \ldots, m\} \tag{5}$$

where each x_j is associated with one of the inter-node links e_j and represents the number of actions for which a source node stores data before transmission. Each coefficient a_{ij} is determined based on the number of data units a source node s_i generates for link e_j. Thus, the set of feasible solutions can be represented by vectors of the form $\mathbf{x} = \{x_1, x_2, \ldots, x_m\}$.

4.2 Problem Complexity

The objective function $w(\mathbf{x})$ is a summation over several convex functions, turning the objective function into a convex separable function. Our convex separable problem has been shown to be tractable under very few conditions. It is solvable in polynomial time, if the set of feasible solution is bounded [16], or constraint matrix is unimodular [17]. None of these conditions apply to our problem. In investigating polynomial time solutions for the buffer assignment, we develop greedy-based algorithms which have less computational complexity than the Integer Programming (IP) techniques.

5. BASIC GREEDY ALGORITHM

The algorithm starts with an initial feasible solution and iterates an augmentation process which moves greedily to-

Algorithm 1 Basic greedy solution for buffer allocation

Require: Optimization problem **P**:(w,F) and initial solution $\mathbf{x}^0 \in F$
Ensure: Solution $\mathbf{x}$ 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 $\mathbf{x}$

ward the final solution. The key idea is to increment one variable x_j during each iteration, which is equivalent to increasing only one inter-node buffer at each time. The minimization objective in (3) enforces the algorithm to evenly increment all variables unless increasing a buffer would violate memory constraints. Since the basic greedy is motivated by several basic concepts mostly used in combinatorial optimization over jump systems [18], we first define these terms as follows.

Definition 1. *For any two lattice points $\boldsymbol{x}$ and $\boldsymbol{y}$ in Z^m, the box spanned by $\boldsymbol{x}$ and $\boldsymbol{y}$ is denoted by [$\boldsymbol{x}$,$\boldsymbol{y}$] and defined as the set*

$$\{\boldsymbol{z} \in \mathrm{Z}^m | \min(x_j, y_j) \le z_j \le \max(x_j, y_j)\} \quad \forall j \in \{1, ..., m\} \tag{6}$$

The spanned box [$\mathbf{x}$,$\mathbf{y}$] contains all the points $\mathbf{z}$ such that

$$d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{y}) = d(\mathbf{x}, \mathbf{y}) \tag{7}$$

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

$$d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_1 = \sum_{j=1}^{m} |x_j - y_j| \tag{8}$$

Definition 2. *For any two points $\boldsymbol{x}$ and $\boldsymbol{y}$ in Z^m, a step s from $\boldsymbol{x}$ to $\boldsymbol{y}$ is a point $\boldsymbol{x}' \in \mathrm{Z}^m$ such that $\boldsymbol{x}' \in [\boldsymbol{x}, \boldsymbol{y}]$ and $d(\boldsymbol{x}, \boldsymbol{x}') = 1$.*

The definition of step implies that in any step from $\mathbf{x}$ to $\mathbf{y}$, 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($\mathbf{x}$+s) as used in [19].

The 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 $\mathbf{x}$. It iterates through subsequent steps as follows. At each time, it checks all components x_j within vector $\mathbf{x}$ and finds the step St($\mathbf{x}$+s) that minimizes the objective function w. It then updates the current vector $\mathbf{x}$ and chooses the next step by setting $\mathbf{x} \leftarrow \mathbf{x} + \hat{s}$. The algorithm terminates when no other feasible solution exists that improve 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 1. *Any step s from a current feasible solution $\boldsymbol{x}^{curr}$ to the final solution $\boldsymbol{x}^{final}$, 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 $\mathbf{x}$. □

Definition 3. *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.*

In addition to the objective function w and feasible solutions F defined by (3)-(5), 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 (5). 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 1) that maintains locally optimum solution. Each link augmentation allocates some buffer to both processing blocks connected by the link as discussed in section 4.

Lemma 2. *Unless memory constraints in (4) 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 through m iterations, but x_l which remains unchanged ($x_l = 1$) and x_p which is increased by 2. Therefore,

$$\begin{aligned} w_1 &= \sum_{j=1}^{m} \frac{1}{x_j + 1} = \frac{m}{2} \\ w_1' &= \frac{1}{x_l} + \frac{1}{x_p} + \sum_{j=1, j\neq l,p}^{m} \frac{1}{x_j + 1} = \frac{m}{2} + \frac{5}{6} \end{aligned} \tag{9}$$

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,

$$w_{k+1} = \sum_{j=1}^{m} \frac{1}{x_j + 1} = \frac{m}{k+2} \tag{10}$$

$$\begin{aligned} w_{k+1}' &= \frac{1}{x_l} + \frac{1}{x_p} + \sum_{j=1, j\neq l,p}^{m} \frac{1}{x_j + 1} \\ &= \frac{m}{k+2} + \frac{2}{(k+1)(k+2)(k+3)} \end{aligned} \tag{11}$$

Since $w_{k+1} < w_{k+1}'$ the algorithm will augment all internode links evenly in any round of the algorithm. □

Theorem 3. *The greedy algorithm for a given buffer allocation problem* $\boldsymbol{P}$ *as in (3)-(5) is polynomial in the number of dependency links* m *and the size of memory* M*.*

PROOF. Assume the algorithm starts with an initial solution $\mathbf{x}^0$. It takes steps that minimize the objective function in (3). Base on Lemma 2, 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

$$\Delta_i = \sum_{j=1}^{m} a_{ij} \tag{12}$$

where a_{ij} are given by the constraints in (4). The remaining memory (M-W-Δ_i) will be further assigned to different buffers in subsequent steps. When increasing a variable x_j violates constraints, the algorithm ignores that variable from further buffer allocation. The number of such rounds differs from one variable to another. However, in worst case, a link may require $\frac{M-W}{\Delta_i}$ rounds before running out of memory. Because each round of the algorithm takes at most m steps, the maximum number of iteration can be calculated as $m \times \frac{M-W}{\Delta_i}$. Assuming fixed W and Δ_i for a given problem, the total number of steps $\mathrm{St}(\mathbf{x}+s)$ required for the algorithm to converge is $O(m \times M)$ □

6. DISTRIBUTED ALGORITHM

The distributed algorithm has two advantages over the basic greedy: 1) The process of finding optimal buffer sizes is distributed among sensor nodes. 2) The algorithm takes multiple steps at each phase, reducing the total runtime of the buffer allocation. In the distributed scenario, each node assigns inter-node buffers for the entire network, and one of the nodes propagates its local results to other nodes. On receiving this information, other nodes will update buffer assignment, exclude saturated links from the local computation and repeat the process. Although all the nodes make a local buffer allocation, only one node communicates its buffer assignment at each round of the algorithm. Such node should meet an important criterion: the buffer allocation made by the node needs to satisfy memory constraints at all the nodes. In fact, any buffer assignment sent to other nodes is required to be valid for entire network. Therefore, the node that is most constrained in terms of memory will initiate communication because it is least likely to violate the memory constraints in the nodes.

Each node s_i maintains a timer variable τ_i which denotes the time left for the node to initiate communication. It also keeps track of saturated links by a variable *Saturated Links Vector* (SLV_i) of length m whose elements are initially set all to 0 indicating that all the links are unsaturated at the startup. When a link becomes saturated, the corresponding location in every SLV_i is set to 1. Furthermore, each node has a *Buffer Assignment Vector* (BAV_i) which represents current values v_{ij} assigned by node s_i for links x_j. In addition, node s_i maintains a variable $\tilde{M}_i$ that shows the remaining memory and is originally equal to $M_i - W_i$ as given by (4). The key idea behind this algorithm is to increase all buffer sizes associated with unsaturated links by a fixed factor of unit buffer size such that memory constraints are satisfied on all the nodes. The algorithm takes several steps as follows.

Step 1 (Initialization): Each sensor node s_i sets its vector SLV_i to (0, 0, ..., 0), and sets BAV_i to (0, 0, ..., 0). Also, $\tilde{M}_i$ is set to $M_i - W_i$.

Step 2 (Buffer Allocation): For unsaturated links $x_1, x_2, \ldots, x_l$ with current values $v_{i1}, v_{i2}, \ldots, v_{il}$, the node s_i increases v_{ij} by $\hat{v}_i$ as given by

$$\hat{v}_i = \left\lfloor \frac{\Delta_i}{\tilde{M}_i} \right\rfloor \tag{13}$$

where Δ_i is given by (12) and is equivalent to the amount of consumed memory at node s_i due to augmenting all unsaturated links. In fact, $\hat{v}_i$ in (13) defines the maximum number of times all links can be incremented without violating constraints at s_i. The process of buffer allocation may saturate some of the links adjacent to node s_i. For such links, node s_i updates the vector SLV_i by changing the corresponding elements to 1. Once locally maximal $\hat{v}_i$ is selected at each s_i, the algorithm needs to select the node whose local decision contributes the most to the convergence of the algorithm without violating any of the local constraints. Lemma 6 shows that the node with the smallest $\hat{v}_i$ is the only one that is guaranteed to satisfy all of the local constraints. Therefore, to satisfy local constraints each node s_i sets a timer τ_i to be proportional to $\hat{v}_i$. The value of this timer is given by

$$\tau_i \propto \hat{v}_i = \left\lfloor \frac{\Delta_i}{\tilde{M}_i} \right\rfloor \tag{14}$$

Step 3 (Transmission): When the value of the timer τ_i becomes zero, the node s_i broadcasts its local statistics including SLV_i and BAV_i. If all links adjacent to this node are saturated, s_i can stop participating in the buffer allocation process.

Step 4 (Update): On receiving data, each node s_k first terminates its timer to avoid the scheduled transmission. It then updates its local statistics SLV_k and BAV_k by replacing them with the received vectors SLV_i and BAV_i from node s_i.

Step 5 (Termination): After updating local statistics, each node s_k checks if any unsaturated links remain. If all the links are saturated, it declares a termination and reports the BAV_k as final solution. Otherwise, the algorithm proceeds through steps 2, 3, 4 and 5 until it terminates.

Lemma 4. *The buffer allocation performed by a node* s_i *during distributed greedy algorithm saturates at least one of the links.*

PROOF. Assume BAV_i=($v_{i1}, v_{i2}, \ldots, v_{il}$) before the node s_i updates buffer sizes. The node s_i increases each value v_{ij} by $\hat{v}_i$ as given by (13), setting v_{ij} to $v_{ij}+\hat{v}_i$. Further, assume none of the links x_j are saturated due to buffer assignment. Therefore, it is possible to further augment each of the links and set their corresponding values to $v_{ij}+\hat{v}_i+1$, which contradicts (13). Thus, at least one of the links gets saturated during each buffer allocation. □

Theorem 5. *The distributed algorithm for a given problem* $\boldsymbol{P}$ *as in (3)-(5) is polynomial in the number of dependency links,* m*.*

PROOF. Based on Lemma 4, during each buffer assignment at least one of the dependency links is saturated. There-

fore, the algorithm will require at most m rounds to terminate. □

Lemma 6. *Only the buffer assignment introduced by the node s_i with smallest timer value τ_i guarantees that memory constraints of all the nodes are satisfied.*

PROOF. Without loss of generality, node s_i with timer τ_i and current available memory $\tilde{M}_i$ be the node that initiates communication at some round of the algorithm, and s_k with timer τ_k and remaining memory $\tilde{M}_k$ be the node that receives information from s_i. Obviously, memory constraints hold at s_i itself:

$$\hat{v}_i \times \sum_{j=1}^{l} a_{ij} \leq \tilde{M}_i \tag{15}$$

Since s_i initiated communication, it has smaller timer value: $\tau_i \leq \tau_k$. Therefore,

$$\frac{\tilde{M}_i}{\sum_{j=1}^{l} a_{ij}} \leq \frac{\tilde{M}_k}{\sum_{j=1}^{l} a_{kj}} \tag{16}$$

Equations (13) and (16) give

$$\hat{v}_i \times \sum_{j=1}^{l} a_{kj} \leq \tilde{M}_k \tag{17}$$

which means the solution $\tilde{v}_i$ provided by s_i would meet constraints at node s_k. However, if $\tau_i \leq \tau_k$ relation does not hold for all k, (17) and (16) are not guaranteed to hold. □

Theorem 7. *The distributed buffer allocation for a given problem $\boldsymbol{P}$ as in (3)-(5) produces the same results as given by the centralized algorithm described in Algorithm 1.*

PROOF. Let $\mathbf{x}^{final}$ be the final solution produced by the centralized algorithm. Each round of the distributed algorithm that is conducted by a node s_i will augment all unsaturated links by increasing variable v_{ij} by $\hat{v}_i$. The new vector obtained by this round is essentially within the box $[\mathbf{x}^0,\mathbf{x}^{final}]$ (see Lemma 2). Therefore, the final solution by the distributed algorithm is within the above spanned box. Every point within this box is a feasible solution, however, only $\mathbf{x}^{final}$ saturates all the links. Since the final solution by the distributed algorithm saturates all dependency links, it is equivalent to $\mathbf{x}^{final}$. □

7. DYNAMIC RECONFIGURATION

As activity-related context information becomes available to the system, the communication model may change in order to satisfy new dependencies between the nodes. This information may infer new movements or subject's location which can be used to restrict upcoming actions to a subset of possible movements. The goal of dynamic buffer allocation is to reassign buffers based on changes in the 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. Our dynamic reconfiguration scheme is a simple and effective approach for dynamic assignment of the buffers based on the previously presented greedy solutions. Deletion of an inter-node link makes some memory available which can be used to assign buffers to the remaining links. However, similar static effectiveness of the solution is not guaranteed if only memory of the deleted links is taken into consideration when assigning buffers to the new links. To overcome this problem, we explore certain properties of the greedy approach which allow us to dramatically 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 introduced by the static approach.

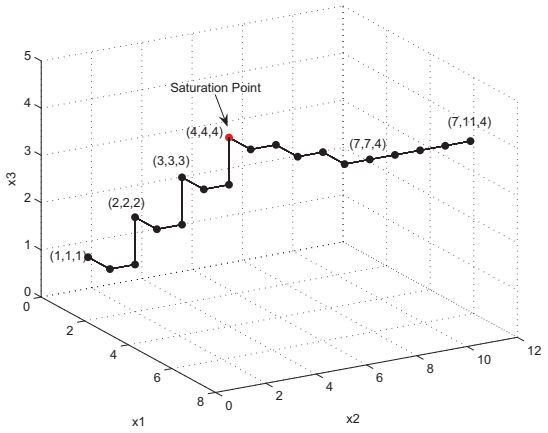


Figure 3: Illustration of saturation point.

Definition 4. *Given problem $\boldsymbol{P}$ with objective function w as in (3) and feasible solutions F in (4)-(5), and initial solution $\boldsymbol{x}^0 = (1, 1, \ldots, 1)$, a saturation point is a point $\boldsymbol{x}^{sat}$ within spanned box $[\boldsymbol{x}^0,\boldsymbol{x}]$ ($\boldsymbol{x} \in F$) such that all components of $\boldsymbol{x}^{sat}$ have the same value v^{sat}, and v^{sat} is maximized.*

$$\boldsymbol{x}^{sat} = (v_1, \ldots, v_m) \quad \forall j \;\; v_j = v^{sat} \tag{18}$$

$$v^{sat} = \arg\max_{v}(v_1, \ldots, v_m) \in F \quad \forall j \;\; v_j = v \tag{19}$$

When the greedy algorithm iterates through steps, different links are saturated one after another. Once a link is saturated, it is ignored in the next iterations. The saturation point, in fact, refers to the first link that is eliminated from further augmentation. Figure 3 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 iterates 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.

Lemma 8. *Given final solution $\boldsymbol{x}$=(v_1, …, v_m) obtained from the greedy algorithm, the saturation point $\boldsymbol{x}^{sat}$ = (v^{sat}, …, v^{sat}) is calculated by*

$$v^{sat} = \min(v_1, \ldots, v_m) \tag{20}$$

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 3). □

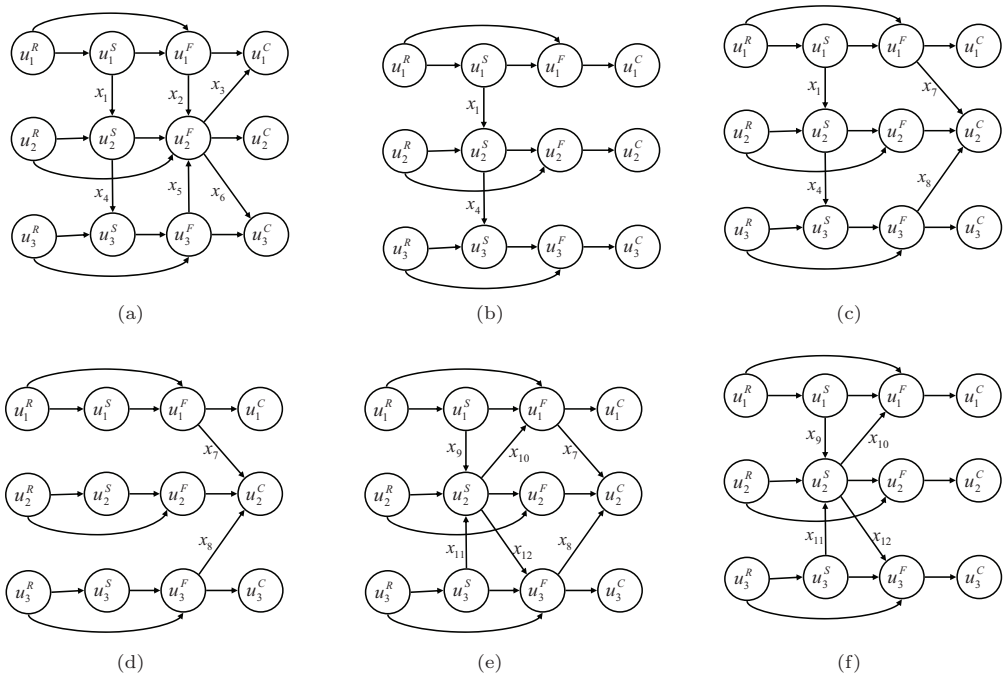


Figure 4: Task graphs for a network of three sensor nodes with dynamic reconstruction of inter-node links.

7.1 Link Deletion

Removing an existing link will create free space for other links to be augmented. On deletion of a link, the system runs the 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 9. *Given a buffer allocation problem $\boldsymbol{P}$, any point in the box spanned by two feasible solutions $\boldsymbol{x}$ and $\boldsymbol{x}'$ remains a feasible solution if any memory size M_i increases.*

PROOF. Let $\mathbf{P}$ 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 $\mathbf{P}$ 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 (4). Therefore, the new expanded box contains all previous feasible solutions. In particular, it contains all the points within the box $[\mathbf{x},\mathbf{x}']$. □

Theorem 10. *By removing a link from dependency graph, the greedy algorithm with initial solution $\boldsymbol{x}^{sat}$ gives the same result as that with the initial solution $\boldsymbol{x}^0$ 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 $\mathbf{x}=(x_1, \ldots, x_{m-1})$. Let w and w' be the values of the objective function using initial solutions $\mathbf{x}^0=(1, \ldots, 1)$ and $\mathbf{x}^{sat}=(v^{sat}, \ldots, v^{sat})$ respectively. Because link deletion makes some memory available, any point within the box $[\mathbf{x}^0,\mathbf{x}^{sat}]$ is a feasible solution for the problem (see Lemma 9). Thus, $\mathbf{x}^{sat}$ could be an initial point for the greedy algorithm. Based on Lemma 2, 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 result the same solutions. □

7.2 Link Insertion

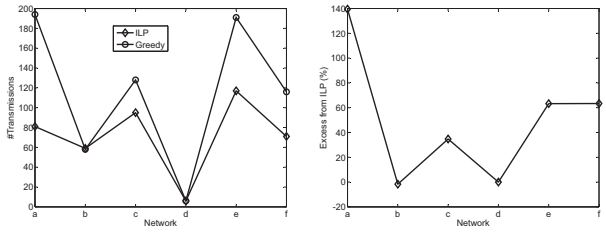
When a new link is added to the network, we still use the saturation point to restart the algorithm. However, this point essentially would not be a feasible solution for the new problem with extra link. The reason for this is that the new variable added to the constraints in (4) may shrink the feasible solution space. Yet, we are interested in finding an initial point whose all components have the same value. This point would essentially 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 done for the static case.

8. EXPERIMENTAL VERIFICATION

In this section, we demonstrate the performance of our buffer assignment techniques through several network configurations.

8.1 Setup

Our experiments are conducted with homogenous sensor nodes, each having a TelosB [20] mote with a memory of size 10 KB. The motes are equipped with a Chipcon CC2420 ra-



(a) # of transmissions (b) Excess from ILP

Figure 5: Comparing ILP and Greedy solutions.

dio (IEEE 802.15.4 standard) for communication. The system is primarily used for movement monitoring applications. For that, we assume that human actions follow the Poisson distribution over a given time period T [21]. The probability of observing x number of actions for given time interval T is represented by $p(x)$. Let λ be the expected number of actions occurred with a confidence level of $1-\alpha$. The Poisson model provides an upper bound k for the action rate during an interval.

$$p(x \leq k) = \sum_{x=0}^{k} p(x) = \sum_{x=0}^{k} \frac{\lambda^x e^{-\lambda}}{x!} = 1 - \alpha \qquad (21)$$

The action rate, k, is calculated based on the Poisson distribution model to achieve certain confidence level. This action rate is further used to calculate number of actions A occurred during time period T.

$$A = k \times T \qquad (22)$$

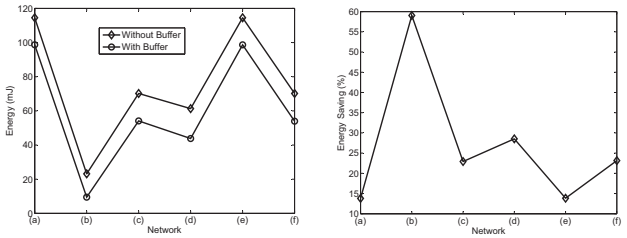
In our analysis, we assume that the actions occur at a rate of 1 action/min and each spans 200 samples. Using these assumptions, we achieve a high confidence level (95%) with the Poisson model. We further assume that the size of intra-node buffer is 2 bytes for segmentation (one for start and one for end of the action), 10 bytes for feature extraction (10 statistical features will be extracted by individual nodes), and 1 byte for classification (each classifier produces a label associated with the current action occurred in the system).

8.2 ILP vs. Greedy

To compare the results obtained by our greedy algorithm with the lower bounds reported by the ILP-based approach, we use the six the task graphs shown in Figure 4. The network has three sensor nodes and changes dynamically. The initial configuration shown in Figure 4(a) has six inter-node dependency links (labeled as x_1 to x_6) for segmentation, feature extraction, and classification. Node 1 receives well-pronounced signals that allow it to perform segmentation independent of the two other nodes. However, node 2 and node 3 can properly segment their signals contingent receiving information from the first and second nodes respectively. Moreover, this model takes into account dependency of features at node 2 on those of the two other nodes. That is, the second node takes the features extracted by the other nodes and performs data fusion at the feature level for classification. Finally, nodes 1 and 3 perform a partial data fusion by combining their local features with those of the second node. Figure 4(b) thru Figure 4(f) illustrate dynamic changes in the network due to potential requirements of the application.

Table 1: Transmission saving.

Conf.	ILP Lower Bound	Greedy
a	98.13%	86.5%
b	98.63%	95.9%
c	97.80%	91.1%
d	99.86%	99.5%
e	97.29%	86.7%
f	98.36%	91.9%
Avg	98.34%	91.93%



(a) Energy Consumption (b) Energy Saving with Buffers

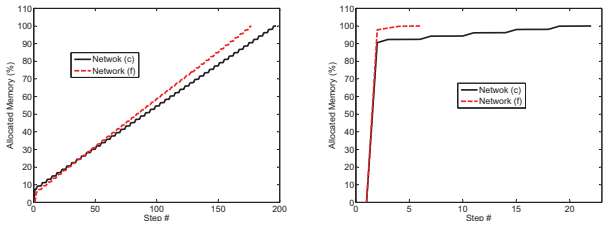
Figure 6: Energy consumption (a) and percentage of energy saving (b) for different networks of Figure 4.

We solve the buffer allocation problem using both integer relaxation and greedy algorithms. The IP solver in [22] is used to specify and solve our convex optimization problem. The results produced by the ILP are used as a lower bound on the solution to demonstrate performance of the greedy algorithms. In Figure 5, we compare the results obtained by our greedy algorithm with the optimal results reported by the ILP-based approach. Figure 5(a) shows the number of transmission achieved by each of the ILP and greedy approaches. Figure 5(b) shows the amount of excess by the greedy solution from the lower bound ILP. The average percentage of excess from the near-optimal values is 49.8%. An interesting observation is that for Figure 4(b) the ILP solver outputs a larger number of packets than the greedy approach (59 versus 58). This is, in fact, due to integer relaxation of the variables x_j when using convex solver. Also, for network shown in Figure 4(d), our greedy obtains the same results as the ILP.

8.3 Energy Saving

In the absence of buffers, each node is required to transmit its local data to other nodes on occurrence of every action. With the assumption of 1 action per minute, this results 1440 transmissions per node for twenty-four hours of system operation. For the networks illustrated in Figure 4 the total number of transmissions in the unbuffered case is 4320 (1 packet every minute for 24 hours). As mentioned previously, Figure 5(a) illustrates number of transmissions while buffer allocation is employed. This would results in an average of more than 91% reduction in number of transmissions as shown in Table 1. Given the fact that the results obtained by the ILP-based approach are near-optimal, our greedy solutions can still reduce the number of transmissions significantly. Table 1 shows percentage of transmission reduction for different algorithms compared with an unbuffered system. On average, the system gains 98% and 91% savings in packet transmission when ILP and greedy algorithms are employed respectively.

We further calculate the amount of energy consumption



(a) Static (b) Dynamic

Figure 7: Allocated memory versus step number for static and dynamic buffer allocation approaches.

Table 2: Number of steps and speedup.

Conf.	Static	Dynamic	Speedup (%)
a	261	-	-
a → b	98	12	87.7
b → c	176	4	97.7
c → d	1002	914	8.7
d → e	266	2	99.2
e → f	196	20	89.8
Avg	333	190	76.6

for both cases of unbuffered and buffered systems. The energy consumption is calculated using the graph shown in Figure 1(b) as a function of the amount of data being transmitted across the network. The results are shown in Figure 6. For a system without any buffers, energy consumption ranges from 23.15 to 114.5 mJ with an average of 75.6 mJ. However, when buffers are used for transmission reduction, the energy consumption ranges between 9.5 and 98.7 mJ with an average of 59.8 mJ. Figure 6(b) shows percentage of energy saving obtained by allocating buffers. Energy saving ranges from 13.8% to 59.0% with an average saving of 26.8%.

8.4 Dynamic Buffer Allocation

Table 2 shows computational complexity of different buffer allocation schemes in terms of the number of steps. The table describes the number of steps required to reconfigure system buffers in case of a topology change. The required number of steps is proportional to the weight of the link that has been modified. For example, "c → d" is relatively costly for both static and dynamic approaches because (d) lacks two heavy links present in (c). When a heavy link is removed - a large amount of memory is freed, which takes a large number of steps to redistribute. The system is expected to gain large computational saving when dynamic buffer allocation is utilized. However, in certain cases, the amount of saving might be small (e.g. network (d)). Figure 7(a) and Figure 7(b) support the same idea. While in the case of static approach in Figure 7(a) memory usage is uniformly increasing, dynamic approach in Figure 7(b) makes a large leap toward the solution and then slowly converge to the final answer.

8.5 Sway Analysis

An interesting application with inter-node dependencies is detection of upper body sway during walking which can be used for assessment of postural control system. In this case, a node placed on the "leg" can segment walking data into gait cycles while a node placed on the "neck" cannot recognize the start and stop times of the gait cycle. The "neck"

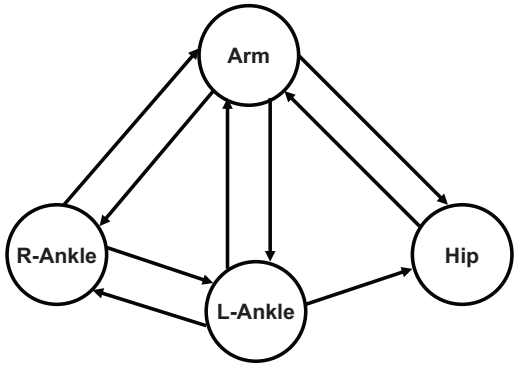


Figure 8: Dependencies for action recognition and sway analysis.

Table 3: Data units produced by the nodes for sway analysis.

Dependency	Data Rate (bytes/action)
R-Ankle → Arm	2 (segmentation) & 1 (classification)
L-Ankle → Arm	2 (segmentation) & 1 (classification)
Hip → Arm	2 (segmentation) & 1 (classification)
Arm → R-Ankle	2 (segmentation) & 1 (classification)
Arm → L-Ankle	2 (segmentation) & 1 (classification)
Arm → L-Ankle	2 (segmentation) & 1 (classification)
L-Ankle → R-Ankle	5 (gait parameters)
R-Ankle → Hip	5 (gait parameters)

node is capable of measuring lateral sway. Therefore, the "leg" node produces segmentation data that can be used by the "neck" node for sway analysis. The system is composed of four sensor nodes placed on "hip", "arm", "right-ankle" and "left-ankle". A detail description of inter-node dependencies for this application can be found in [23]. The "hip" node is responsible for measuring sway. The nodes on the two ankles are used in collaboration to extract gait cycle parameters. When the walking action is detected, one of lower body nodes (e.g. "right-ankle") sends information regarding gait parameters to the "hip" node where a final analysis on quality of stability is performed. Since the system is aimed to assess balance control during walking, it must be able to detect walking first. Therefore, we a node is used on the "arm" to make observations on upper body movements. The "arm" node is assumed to be master for "segmentation" and "classification". Figure 8 illustrates inter-node dependencies required for balance evaluation as described above. For visualization only dependency between nodes are shown; however, links represent dependencies between processing modules within the nodes.

We first perform an analysis on classification accuracy of this BSN model. The system is able to recognize actions with an accuracy of 93.87% with real data collected from three healthy subjects performing 25 transitional movements such as 'sit to stand', 'sit to lie', 'jump', 'kneel' and 'walk'. Table 3 shows the amount of data produced, per action, by each source node to be sent to a destination node. This results in a total of 14 dependency links between the processing units of the four nodes. Human movements can occur at different speeds. We assume a normal walking speed of 110 steps/min. (55 actions/min. or 0.92 Hz) for our gait analysis study [23].

With the aforementioned experimental setup, we run our

Table 4: Performance of buffer allocation for sway analysis.

	W/o Buffer	W/ Buffer	Saving(%)
#Transmissions	5760	428	92.6%
Energy (mJ)	472.6	313.4	33.7%

greedy algorithm to find the size of each buffer on the communication links. Table 4 shows the total number of transmissions and the amount of energy for a system without buffer and a system with buffers. As shown in the table, by employing buffers, 33.7% reduction in energy consumption can be achieved.

9. CONCLUSION

In this paper, we outlined a communication model to reduce the number of transmissions over BSN networks. We presented a simple and efficient buffer assignment algorithm in distributed light-weight sensory systems with limited storage. Our technique maps the communication minimization model into a convex optimization problem and provides combinatorial solutions that calculate buffer sizes associated with each processing module. We compared the performance of our greedy solutions with the lower bounds defined by the ILP and presented the amount of energy saving that can be obtained using our buffer allocation techniques. We further developed a dynamic resource allocation technique based on our greedy approach which reduces time required for buffer allocation 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," *Signal Processing Magazine, IEEE*, vol. 19, no. 2, pp. 40–50, Mar 2002.

[2] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on*, vol. 2, Jan. 2000, p. 10.

[3] 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.

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

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

[6] J. Ammer and J. Rabacy, "The energy-per-useful-bit metric for evaluating and optimizing sensor network physical layers," in *Sensor and Ad Hoc Communications and Networks, 2006. SECON '06. 2006 3rd Annual IEEE Communications Society on*, vol. 2, Sept. 2006, pp. 695–700.

[7] A.-M. Lin and J. Silvester, "Priority queueing strategies and buffer allocation protocols for traffic control at an atm integrated broadband switching system," *Selected Areas in Communications, IEEE Journal on*, vol. 9, no. 9, pp. 1524–1536, Dec 1991.

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

[9] J. Hu, U. Y. Ogras, and R. Marculescu, "System-level buffer allocation for application-specific networks-on-chip router design," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 12, pp. 2919–2933, Dec. 2006.

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

[11] A. Benveniste, B. Caillaud, L. P. Carloni, P. Caspi, A. L. Sangiovanni-Vincentelli, and S. Tripakis, "Communication by sampling in time-sensitive distributed systems," in *EMSOFT '06: Proceedings of the 6th ACM & IEEE International conference on Embedded software*, 2006, pp. 152–160.

[12] 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: Proceedings of the 7th ACM & IEEE international conference on Embedded software*. New York, NY, USA: ACM, 2007, pp. 231–239.

[13] S. Tripakis, C. Pinello, A. Benveniste, A. Sangiovanni-Vincent, P. Caspi, and M. Di Natale, "Implementing synchronous models on loosely time triggered architectures," *IEEE Trans. Comput.*, vol. 57, no. 10, pp. 1300–1314, 2008.

[14] C. Sofronis, S. Tripakis, and P. Caspi, "A memory-optimal buffering protocol for preservation of synchronous semantics under preemptive scheduling," in *EMSOFT '06: Proceedings of the 6th ACM & IEEE International conference on Embedded software*, 2006, pp. 21–33.

[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] S. M. Stefanov, "Convex separable minimization problems with a linear constraint and bounded variables," *International Journal of Mathematics and Mathematical Sciences*, vol. 2005, no. 9, pp. 1339–1363, 2005.

[17] 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.

[18] A. Bouchet and W. H. Cunningham, "Delta-matroids, jump systems, and bisubmodular polyhedra," *SIAM J. Discret. Math.*, vol. 8, no. 1, pp. 17–32, 1995.

[19] 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.

[20] J. Polastre, R. Szewczyk, and D. Culler, "Telos: enabling ultra-low power wireless research," *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pp. 364–369, April 2005.

[21] 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.

[22] 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.

[23] H. Ghasemzadeh and R. Jafari, "Data aggregation in body sensor networks: A power optimization technique for collaborative signal processing," in *Sensor Mesh and Ad Hoc Communications and Networks (SECON), 2010 7th Annual IEEE Communications Society Conference on*, 21-25 2010, pp. 1 –9.