Energy Management for Time-Critical Energy Harvesting Wireless Sensor Networks

Bo Zhang, Robert Simon, and Hakan Aydin

Department of Computer Science
George Mason University, Fairfax, VA 22030
{bzhang3,simon,aydin}@cs.gmu.edu

Abstract. As Cyber-Physical Systems (CPSs) evolve they will be increasingly relied on to support time-critical monitoring and control activities. Further, many CPSs that utilize Wireless Sensor Networking (WSN) technologies require energy harvesting methods to extend their lifetimes. For this important system class, there are currently no effective approaches that balance system lifetime with system performance under both normal and emergency situations. To address this problem, we present a set of Harvesting Aware Speed Selection (HASS) algorithms. We use an epoch-based architecture to dynamically adjust CPU frequencies and radio transmit speeds of sensor nodes, hence regulate their power consumption. The objective is to maximize the minimum energy reserve over any node in the network, while meeting application's end-to-end deadlines. Through this objective we ensures highly resilient performance under emergency or fault-driven situation. Through extensive simulations, we show that our algorithms yield significantly higher energy reserves than the approaches without speed and power control.

1 Introduction

There is an increasing need to effectively support Wireless Sensor Network applications that have significant data collection and processing requirements. Examples range from Wireless Network Video Systems for surveillance [19] to Cyber-Physical Systems such as smart power grid using 802.15.4/Zigbee technology [14]. These types of systems often have strict timing and performance specifications. For instance, smart power grid systems need to provide real-time pricing information, while water distribution systems need to instantly react to a contamination. Further, many of these self$-^*$, unattended and deeply-embedded systems will be expected to last for several decades, and therefore must carefully manage available energy resources. The challenge faced by system designers is to balance the performance and system availability requirements with energy management policies that can maximize system lifetime.

One approach for maximizing system lifetime is to use energy harvesting [6]. By harvesting energy from environmental sources such as solar, wind or water flow, WSN nodes potentially have perpetual energy supply. However, given the large energy demands of the computational and communication intensive

WSN applications, and limited availability of environmental power, perpetual operation of WSN nodes cannot be realized without deliberate energy management. This problem is exacerbated if the application has unpredictable spikes in workload demand, such as a water distribution system reacting to a biological contamination. *The focus of this paper is a coordinated energy management policy for time-critical WSN applications that use energy harvesting and that must maintain required performance under emergency or fault-driven situations.*

Our approach is to make combined use of two energy saving techniques, Dynamic Voltage Scaling (DVS) [2] and Dynamic Modulation Scaling (DMS) [18]. The DVS technique saves computation energy by simultaneously reducing CPU supply voltage and frequency. The DMS technique saves communication energy by reducing radio modulation level and hence transmit speed. To take advantage of these methods we propose a set of *Harvesting Aware Speed Selection* (*HASS*) algorithms that use both DVS and DMS in conjunction with energy harvesting modules. The purpose of the *HASS* approach is to maximize energy reserves while meeting application performance requirements, therefore maximize the system's resilience to emergency situations.

One difficulty in managing energy for these systems is that nodes may have quite different workload demands and available energy sources. This may arise from natural factors such as differences in nodes energy harvesting opportunities, or unbalanced distribution of processing workloads or network traffic. Because of these conflicting design considerations, the *HASS* approach attempts to maximize the minimum energy reserve level over any node in the network, while guaranteeing required system performance. Specifically, *HASS* adjusts CPU processing speed and radio transmit speed, with a goal that the harvesting-rich nodes run faster to allow the harvesting-poor nodes to slow down and save energy, given tight end to end data collection latency constraint. The reduced energy consumption will secure higher energy reserves for the poor nodes.

Our specific contributions are summarized as follows: We first provide a basic architectural description for DVS and DMS nodes that use energy harvesting. We then propose a general network and performance model for time-critical WSN applications. Unlike the majority of existing works in energy harvesting WSN systems which mainly focus on individual nodes, we target a multi-hop sensor network with an end-to-end performance requirement. Next, we show how to formulate the problem of maximizing the minimum energy reserve while maintaining required performance as an optimization problem. We prove that this problem can be solved optimally and efficiently. We also propose and evaluate both centralized and distributed protocols to implement the *HASS* solution. We conducted extensive simulations to evaluate our methods under a variety of processing, communication and performance requirements. Unlike most existing works which assuming solar energy as the environmental sources in their simulations, we propose an experimental methodology to simulate energy harvesting WSN systems utilizing energy harvested from water flow in a water distribution system. Our results show that both centralized and distributed solutions significantly improve the capacity of time-critical WSN systems to deal with emergency situations, in addition to meeting performance requirements.

2 Background and Related Work

The joint use of DVS and DMS in wireless embedded systems has been explored in [7] and [17]. In [7], Kumar et al. addressed a resource allocation problem with the aim of minimizing energy consumption. They assume a system containing a mixed set of computation and communication tasks. In [17], the energy management problem is formulated as a convex optimization problem, which is then addressed through the use of genetic algorithms. In [18], Yu et al. proposed DMS-based approach for a multi-hop WSNs. They assume a data collection application in which a base station periodically collects sensed data from WSN nodes over a tree-based routing structure. In [21], we proposed a joint DVS-DMS energy management approach for individual energy harvesting WSN nodes with a goal of maximizing the minimum energy level over time. Unlike our work, [7], [17], [18] assume battery-powered system and target prolonging system lifetime by reducing energy consumption, without considering the need of ensuring perpetual operation through energy harvesting.

Many existing studies explored the design of energy harvesting WSNs. In [10], Moser et al. proposed the LSA algorithm (Lazy Scheduling Algorithm) for scheduling real-time tasks in the context of energy harvesting. LSA defers task execution and hence energy consumption as late as possible so as to reduce the amount of deadline misses. Liu et al. ([8]) proposed EA-DVFS (Energy-Aware Dynamic Voltage and Frequency Scaling) which improves the energy efficiency of LSA by using DVS. Both LSA and EA-DVFS manage only the CPU energy, while ignoring radio energy. Other related work includes [5] and [12] which aim at balancing energy supply and energy demand in energy harvesting WSN systems. Finally, [6], [11] proposed to maximally utilize harvested energy for maximizing the amount of completed works, and hence system performance. Neither of these works considered maximizing minimum energy level by using joint DVS-DMS techniques.

3 System Architecture

This section describes our architecture for energy harvesting WSN systems supporting time-critical applications. It consists of a basic node and device model, a task-based workload model and energy consumption analysis, and a performance model. This will provide a systematic methodology for modeling and analyzing the performance of this type of systems.

3.1 Device Model

Without loss of generality, we assume that each node has several functional units, including an energy harvester head, an energy storage unit, a DVS capable

CPU, a DMS capable radio, as well as required sensor suites. The harvester head is energy source-specific, such as solar panel or wind generator. The energy storage unit (e.g. rechargable battery or super-capacitor) has a maximum energy capacity of Γ^{max} joules. This unit receives power from the energy harvester, and delivers power to the sensor node. We take the commonly used approach that the amount of harvested power is uncontrollable, but reasonably predictable, based on the source type and harvesting history [6]. To capture the time-varying nature of environmental energy, time is divided into *epochs* of length S. Harvested power is modeled as an epoch-varying function denoted by P_i^h, where i is the epoch sequence number. P_i^h remains constant within the course of each epoch i, but changes for different epochs. To be precise, P_i^h is the actual power received by energy storage which incorporating the loss during power transfer from energy harvester to energy storage, and the power leakage of energy storage. The time unit used for harvesting prediction is therefore one epoch. The *prediction horizon*, H is an interval containing a number of epochs during which predictions can be reasonably made. Our approach needs to know the harvested power prediction of only the coming epoch, at the epoch start.

The node consumes power via either processing, radio communication or sensing. We now describe how to model energy consumption for an individual node. The basic time interval over which energy consumption is calculated is called a *frame*, defined precisely below in Section 3.2. Frames are invoked periodically. We assume the DVS-enabled CPU has m discrete frequencies $f_{min}{=}f_1{<}...{<}f_m{=}f_{max}$ in unit of cycles per second, and the DMS-enabled radio has n discrete modulation levels, $b_{min}{=}b_1{<}...{<}b_n{=}b_{max}$. We use the terms frequency and compute speed interchangeably. In practice, the modulation level represents the number of bits encoded in one signal symbol [18]. To understand this relationship, let R be the fixed symbol rate. Then modulation level b is associated with communicate speed d and expressed as:

$$d = R \cdot b \tag{1}$$

Let e^{sen} represents the energy required for each sensing event. Note that e^{sen} is a constant. The computation energy e_k^{cp} in the k^{th} frame is a function of compute speed f_k and supply voltage $V_{dd,k}$ [2]. The communication energy e_k^{cm} in the k^{th} frame is a function of communicate speed d_k [18]. Then we have:

$$e_k^{cp} = [\alpha f_k V_{dd,k}^2 + P^{ind,cp}] \cdot \frac{C}{f_k} \tag{2}$$

$$e_k^{cm} = [\beta R(2^{d_k/R} - 1) + P^{ind,cm}] \cdot \frac{M}{d_k} \tag{3}$$

Above, C and M are the computation and communication workloads in a frame. C is the number of CPU cycles to be processed, M is the number of bits to be transmitted. The α in Eq. (2) is the CPU switching capacitance which is a constant. The β in Eq. (3) is a constant determined by the transmission quality and noise level [18]. The terms $\alpha f_k V_{dd,k}^2$ and $\beta R(2^{d_k/R} - 1)$ give the *speed-dependent power* of CPU and radio which vary with f_k, $V_{dd,k}$, and d_k respectively.

$P^{ind,cp}$ and $P^{ind,cm}$ are two constants representing the *speed-independent power* of CPU and radio. By using DVS, the supply voltage $V_{dd,k}$ can be reduced linearly alongside with f_k to obtain energy saving (i.e. $f_k \propto V_{dd,k}$), making the speed-dependent CPU power a *cubic* function of f_k. Our model assumes a sufficient level of coordinated sleeping and transmission scheduling, so that the radio energy consumed by listening channel activities is not a significant factor. Finally, the total energy consumed in frame k, e_k^c equals:

$$e_k^c = e^{sen} + e_k^{cp} + e_k^{cm} \tag{4}$$

3.2 Network and Application Model

The system consists of N sensor nodes and the set of wireless links connecting them. A sensor node is denoted as V_i. Base stations or control points are denoted as BS. The N nodes are divided into two types: *source* nodes perform sensing, processing and communication operations, while *relay* nodes only perform processing and communication. Our data processing architecture is quite general, and supports systems that perform some levels of aggregation at each node, as well as systems that do not allow any aggregation. We represent a time-critical and performance sensitive WSN application by requiring all source nodes report their readings, which may or may not be aggregated into other readings, every π time units. The time interval π is the length of a data collection *frame*. Such frame-based data collection mechanism is quite common for WSN applications [7] [13] [18]. In other words, all sensed, processed or aggregated data must reach BS by the end of each frame. For example, at the start of the k^{th} frame (i.e. at time $(k-1)\cdot\pi$), each source node senses the environment and sends sensed data to BS. The data is routed by other nodes and must reach BS by the end of that frame, at time $k \cdot \pi$. We assume all nodes are time-synchronized so that they are aware of the same frame start and end times.

On a per-frame basis, energy consuming activities within each node are represented using a task-based model. In this way, frame-based energy consumption is determined by examining the energy demands of individual tasks (Eq. (4)). There are a total of three task types: *sensing*, *computation* and *communication*. Without loss of generality and in order to simplify the modeling process, we assume the three tasks are executed in the order of *sense*$\rightarrow$*compute*$\rightarrow$*communicate*. That is, in each frame, a node performs sensing first, then processes the sensor reading, then transmits the processed data. The workloads of the computation and communication tasks of any node V_i are fixed over any frame in a given epoch, and denoted as C_i and M_i, respectively.

We assume that each node uses standard WSN energy management techniques for transitioning to *sleep* states when there is no active task. We also assume that compute and communicate speeds only change at the start of an epoch. This design decision reduces the required level of control and synchronization overhead. For instance, the modulation level of a node must not change frequently, since each such change must be conveyed to its receiver in order to ensure correct demodulation of the transmitted data. Using this analysis we can calculate the time required by each node V_i to carry out all activities during frame k, referred as the *per-node latency*, $l_{i,k}$. The per-node latency depends upon the compute speed $f_{i,k}$ and the communicate speed $d_{i,k}$. Then $l_{i,k}$ is given by

$$l_{i,k} = t^{sen} + \frac{C_i}{f_{i,k}} + \frac{M_i}{d_{i,k}} \tag{5}$$

t^{sen} is the sensing time which is a constant. Note that t^{sen} equals zero for relay nodes. We make a common assumption that the effective data transmission time dominates the overall communication time while ignoring the carrier sense time [7], [17], [18]. Thus, the communication time is inversely proportional to $d_{i,k}$.

The system is organized into a data collection and processing tree rooted at BS, using tree construction algorithms such as [1]. In order to support time-critical operation we must define and calculate the *maximum data collection latency* and individual *path latency*. These two values are used in the optimization formulation in Sections 4 and 5 to ensure that all latency requirements are maintained. In each frame, a node V_i receives data from a set of child nodes denoted as $Children(V_i)$. V_i then forwards packets to its parent node, denoted as $Parent(V_i)$, after received data from all its children. Then the maximum data collection latency $L_{tot,k}$ in frame k is the time interval between the start of frame k, and when BS collects all sensed data, given by

$$L_{tot,k} = Max.\{L_{i,k} + l_{i,k} | V_i \in Children(BS)\} \tag{6}$$

Above, $L_{i,k}$ is the latency of the subtree rooted at node V_i, i.e. $L_{i,k} = Max.$ $\{L_{j,k} + l_{j,k} | V_j \in Children(V_i)\}$. The subtree rooted at a leaf node contains only the leaf itself, and hence incurs zero latency.

Next, we define the *path* ρ_i from a node V_i to the root BS as the series of nodes and wireless links connecting V_i and BS. The notation $V_j \in \rho_i$ signifies that V_j is an intermediate node on path ρ_i. The latency $H_{i,k}$ of ρ_i is defined as:

$$H_{i,k} = \sum_{j:V_j \in \rho_i} l_{j,k} \tag{7}$$

Note that by resolving the recursion in Eq. (6), $L_{tot,k}$ actually equals to the latency of the longest path in the tree, i.e. $Max.\{H_{i,k} | \forall \rho_i\}$.

4 Harvesting Aware Speed Selection

Based on the node and network model presented in Section 3, we now formally define the *Harvesting Aware Speed Selection (HASS)* problem. Our goal is to maintain end-to-end performance while maximizing the system's resilience to abnormal or emergency situations. This is accomplished by maximizing the minimum energy level of any node.

The compute and communicate speeds at individual nodes are adjusted at the start of each epoch, and remain fixed throughout that epoch. As defined in

Section 3.1, an epoch is a time interval over which an energy harvesting prediction can be reasonably made. For an arbitrary epoch, the energy consumption $e^c_{i,k}$ and performance latencies $L_{i,k}$, $H_{i,k}$ of node V_i are fixed over any frame k. For simplicity we therefore rewrite them as e^c_i, L_i and H_i. Then the energy level Γ_i of a node V_i at the end of a given epoch is given as:

$$\Gamma_i = \Gamma_i^{init} + P_i^h \cdot S - \lfloor S/\pi \rfloor \cdot e_i^c \tag{8}$$

Γ_i^{init} is the starting energy level of V_i in the epoch. Recall that S is the epoch length. $\lfloor S/\pi \rfloor$ gives the number of frames in an epoch. Using this notation, we define Γ_{min} as

$$\Gamma_{min} = Min\{\Gamma_i | \forall V_i\} \tag{9}$$

Then the goal of our approach is to maximize Γ_{min}. The variables of the problem are the compute and communicate speeds f_i, d_i used by any node V_i in an epoch. Given N nodes in the tree, there are $2N$ unknowns in our problem. The optimal solution to this problem consists of N *speed configurations* (f_i, d_i), one for each node which maximize Γ_{min}. The problem *HASS* is given as:

$$\begin{aligned}
Max \;\; & \Gamma_{min} && (10)\\
s.t. \;\; & \forall \rho_i, H_i \leq \pi && (11)\\
& \forall V_i, f_i \in [f_{min}, f_{max}], d_i \in [d_{min}, d_{max}] && (12)\\
& \forall V_i, 0 < \Gamma_i \leq \Gamma^{max} && (13)
\end{aligned}$$

The constraint (11) ensures that the latency of any path ρ_i in the tree is smaller than the frame period π. As mentioned in Section 3.2, this is equivalent to ensuring that the latency of the entire tree is smaller than π. The constraint (12) gives the available ranges of f and d. The constraint (13) requires that the energy level of any node V_i must be confined to the range $(0, \Gamma^{max}]$. In [6], the authors introduced the *energy neutrality* condition, which essentially states that the energy consumed must be no larger than the energy available, such that Γ_i will never drop to zero. This is a necessary condition for an energy harvesting sensor node to operate non-interruptively and we therefore adopt it as a requirement. The left hand side of constraint (13) (called the *positivity constraint*) must hold in order to ensure energy neutrality, while the right hand side (called the *capacity constraint*) is used to model energy storage capacity. Given known and fixed harvested power, and fixed speeds and power consumption, the variation of energy level also fix throughout an epoch, i.e. either monotonically increase or decrease at a fixed rate. Therefore, ensuring a positive energy level at the end of an epoch also ensures positive energy level at the end of any frame in that epoch.

5 Centralized and Distributed Solutions

This section provides centralized and distributed solutions to problem *HASS*. The centralized version provides an optimal solution, while the distributed version is appropriate for systems that need to avoid single control point.

We first give Lemma 1 which states that solving problem *HASS* with full constraint set is equivalent to solving the same problem but without constraint (13). This enables us to remove constraint (13) and focus on a new problem obtained in this manner, denoted as *HASS-N*. Note that the objective function and all other constraints are retained in *HASS-N*.

Lemma 1. If in the optimal solution to HASS-N, Γ_{min} is strictly positive, then the solution to HASS is identical to that of HASS-N. Otherwise, HASS has no feasible solution.

The proof of Lemma 1 can be found in [20]. In the rest of this paper, we will focus on solving problem *HASS-N*. Solving *HASS-N* requires non-linear optimization methods, since it has a non-linear objective function (Eq. (10)). Such costly methods are difficult to implement on resource-constrained sensor nodes. We will show how to obtain an optimal solution efficiently.

A naive approach to solve *HASS-N* is to exhaustively search over all possible solutions. For a system with N nodes where each node has m compute speeds and n communicate speeds, there are $(mn)^N$ possible solutions, making brute force search impractical. However, we notice that many different solutions yield identical Γ_{min}. Using this observation we can simply enumerate each possible Γ_{min}, check if there exists a feasible solution that yields a minimum energy level (among any node) equaling the enumerated Γ_{min}, while satisfying constraints (11) and (12). The highest Γ_{min} that passes this check is by definition the maximum Γ_{min} that we are looking for.

For each node, mn speed configurations correspond to mn different power consumption levels. Since each node's power consumption is fixed throughout an epoch, a node has exactly mn energy consumption levels over an epoch. Thus, given a known starting energy level and a fixed prediction for how much energy can be harvested, a sensor node could end with at most mn possible energy levels in an epoch. Given N nodes, at the end of an epoch, there could be at most mnN different energy levels in the network, and Γ_{min} can be only of these possible values. The set of possible Γ_{min}s is referred as EL (Energy Level), and has a size of mnN.

5.1 Centralized Version

The centralized *HASS* algorithm is called *CHASS*, and is presented in Algorithm 1. It runs on the base station, and assumes that BS must collect Γ^{init} from each node in the system, and is aware of the available speed configurations of sensor nodes. *CHASS* first computes the possible energy levels of all the nodes using Eq. (8) to build the set EL, then sorts EL in non-increasing order (line 1). *CHASS* proceeds iteratively over the sorted EL starting from the first element (i.e. the highest energy level in EL) (line 2). In each iteration p, it solves a decision problem, called *Feasible Solution* denoted by FS_p, by calling algorithm *Is-Feasible* (line 3). The p^{th} element in EL, $EL[p]$ is input to *Is-Feasible*. The problem FS_p is specified as "Is there a solution which yields Γ_{min}=$EL[p]$, while satisfying constraints (11-12)?"

The loop in line 2-9 iterates through all the elements in EL. It continues if the answer to problem FS_p, ans_p is negative, and terminates once it met a FS_p with positive answer, i.e. in iteration z where FS_z is the first problem encountered with positive answer, $z = Min.\{p \in [1, |EL|] | ans_p = TRUE\}$ (line 4-8). By definition of problem *HASS-N* and *FS*, and the ordering of EL, $EL[z]$ is the maximum Γ_{min} that can be achieved (line 5), while satisfying all the constraints. If *CHASS* proceeds to the end of EL and never received a positive answer to any of the FS_p, this implies problem *HASS-N* has no feasible solution.

The algorithm *Is-Feasible* for solving problem FS_p is given in Algorithm 2. The algorithm has one input, the energy level enumerated in iteration p of *CHASS*, $EL[p]$. It has three returned values, the answer to problem FS_p, ans_p, and two speed sets of length N, F^*, D^* which contain f and d derived for all the nodes in the current iteration. F^*, D^* are returned only if ans is positive, otherwise they are empty.

Algorithm 1. CHASS

```
1. Compute and sort EL (in non-increasing order)
2. for $p = 1$ to $|EL|$ do
3.    $[ans_p, F^*_p, D^*_p]$ = call Is-Feasible($EL[p]$)
4.    if $ans_p$ == TRUE then
5.       $Max\_\Gamma_{min} = EL[p]$
6.       $[F^{opt}, D^{opt}] = [F^*_p, D^*_p]$
7.       Break from for-loop
8.    end if
9. end for
```

Algorithm 2. Is-Feasible - Input: $EL[p]$

```
1. $\Gamma_{min} = EL[p]$
2. for $i = 1$ to N do
3.    $(F^*[i], D^*[i], l_i^{min})$ = call $find\_fastest(\Gamma_{min})$ on $V_i$
4. end for
5. Compute $H_i = \sum_{j:V_j \in \rho_i} l_j^{min}$ for any path $\rho_i$
6. if $\forall \rho_i, H_i \leq \pi$ then
7.    $ans = TRUE$
8. else
9.    $ans = FALSE$, $F^*, D^* = \emptyset$
10. end if
11. return $[ans, F^*, D^*]$
```

We now demonstrate that Algorithm 2 is correct. First, by making $\Gamma_{min} = EL[p]$ (line 1), $\Gamma_i \geq \Gamma_{min} = EL[p]$ must hold for any node V_i. Then the algorithm calls function *find_fastest* for each node (line 2-4) to search over all its *mn* speed configurations for the fastest one, while yielding $\Gamma_i \geq EL[p]$. Specifically, *find_fastest* returns a speed configuration for V_i, $(F^*[i], D^*[i])$ which satisfies:

$$F^*[i] \in [f_{min}, f_{max}], D^*[i] \in [d_{min}, d_{max}] \tag{14}$$

$$\Gamma_i(F^*[i], D^*[i]) \geq EL[p] \tag{15}$$

$$\forall (f, d), l_i(F^*[i], D^*[i]) \leq l_i(f, d) \tag{16}$$

$\Gamma_i(f, d)$ and $l_i(f, d)$ represent the energy level and per-node latency achieved using speed configuration (f, d). *find_fastest* also returns the per-node latency l_i^{min} at V_i achieved by using the derived $(F^*[i], D^*[i])$. Note that l_i^{min} is the least achievable latency according to Eq. (16). Next, for each path ρ_i, we compute its latency H_i by summing up any $l_j^{min}, V_j \in \rho_i$ (line 5). Since (F^*, D^*) minimizes the per-node latency at any node, it also minimizes the latency of any path H_i. Therefore, if $H_i \leq \pi, \forall \rho_i$, the constraint (11) is met, then the answer to problem FS_p is positive (line 6-7). Otherwise, constraint (11) can never be met, hence the answer is negative (line 8-9). Note that it is possible that function $find\_fastest$ does not return an answer, as there may exist some nodes having no possible energy level larger than the input $EL[p]$. In this case, the algorithm immediately rejects $EL[p]$. The speed sets F^*, D^* found in iteration z is set to be the optimal solution to problem *HASS-N* and also *HASS* (line 6 in Algorithm 1). $EL[z]$ is set to be the maximum achievable Γ_{min} (line 5 in Algorithm 1).

It is possible to reduce the runtime of *CHASS* by implementing the search for FS_z in a binary search fashion. This will reduce the number of iterations in *CHASS* from $O(|EL|)$ to $O(\log(|EL|))$. We describe a faster algorithm $CHASS^*$ implemented in this manner and give a complexity analysis in [20].

5.2 Distributed Version

We next describe the distributed *HASS* solution called *DHASS*. The purpose of the distributed version is to enable any node in the network to act as the base station, and therefore enable that node to make command and decisions.

The algorithm *DHASS* proceeds also in binary-search fashion. It requires one initialization round during which each sensor node sends an initialization message containing two pieces of information, its estimated lowest and highest energy levels at the end of the epoch, denoted as Γ^{low} and Γ^{high}. After the initialization round, all the nodes agree on the global lowest and highest achievable energy levels (among the entire tree). The continuous range between the two energy levels is the starting binary search space. Then, it runs for Y computation rounds, each of which corresponds to one iteration of binary search, and solves one problem FS using the distributed *Is-Feasible*. The distributed *Is-Feasible* requires accumulative collection of nodes' latency values, which are in turn used for the root to calculate the end-to-end latency L_{tot}. The root then compares L_{tot} to π in order to determine the answer to problem FS, and disseminates it to all the nodes. Note that any node in the network can be the root. The specification of *DHASS* is given in [20]. Though *DHASS* is only a heuristic-based solution, we demonstrate in [20] that it can closely match the performance of *CHASS*.

6 Performance Evaluation

We performed a series of simulations to evaluate the effectiveness of our *HASS* approaches. The goal of the evaluation is to determine how well both the *CHASS* and *DHASS* algorithm maximize the minimum energy level across the system. The evaluation examined a number of workload scenarios, including several emergency scenarios where there are sudden, unexpected peaks in the demand.

6.1 Experimental Methodology

We evaluated our approaches within a WSN system designed for residential monitoring of water usage and quality. Each customer (residence) is coupled to a supply pipe through a water meter. Each water meter is coupled with a DVS-DMS enabled node. Energy is harvested from the flow of water, using a device such as the one described in [15]. The amount of harvested energy is therefore dependent upon the rate at which the customer uses water. To our best knowledge, we are the first to simulate energy harvesting WSN systems utilizing water flow as the energy source.

We have developed simulation software combining TOSSIM, the standard WSN simulator, with EPANET [9], a public domain, water distribution system modeling program developed by the US Environmental Protection Agency. Our simulator can take as input a variety of WSN topologies, water distribution system configurations and customer usage patterns. Based on water utilization and water quality patterns, the software simulates energy harvesting, and various WSN processing and communication activities. The presented results are based upon a 100 node residential water distribution topology. The topology is derived from an existing suburban area of 100 houses. The 100 nodes installed in the water meters then form a WSN system. We use the *Collection Tree Protocol* [1] to organize the nodes into a data collection tree.

Due to standard repetitive water usage patterns we use a 6 hours cycle, as specified in EPANET. We fix the harvesting horizon at H=6 hours. A horizon is then divided into 24 epochs with equal length S=15 minutes. We run EPANET for 48 hours, containing 8 horizons or equivalently 192 epochs, and obtained hydraulic simulation reports. Using these reports we generate water energy harvesting profile for each node based on the observed water usage at the customer. Note that the difference in the amount of water used across residences will lead to quite different power harvesting profiles across nodes. In [20], we plotted the harvesting profile of one selected node. The frame period is set to π=240ms.

Both algorithm *CHASS* and *DHASS* were implemented in our simulation environment. Although there are no schemes that are directly comparable to our algorithms, we implemented a baseline scheme called No-Power-Management (*NPM*). Unlike the *HASS* approaches, *NPM* scheme is harvesting-unaware in the sense that it uses the highest frequency and modulation level for all the nodes in order to guarantee data collection timing constraint. Our experiments considered two basic application types: applications that support *complete-aggregation* and applications that do not require any aggregation (*non-aggregation*). By complete-aggregation, we mean that each node aggregates multiple packets received into one single packet, while in the non-aggregation case a sensor node forwards all packets to its parent without aggregation. The packet size is randomly selected between M=[64, 128] bytes, and the computational workload is randomly selected between C=[0, 3000000] cycles.

The hardware basis for a DVS-DMS capable platform is the widely available iMote-2 sensor node [16]. The iMote-2 platform has a Intel Xscale PXA27x CPU [4] and a ChipCon CC2420 radio [3]. PXA27x CPU has 6 frequency and power levels as specified in [4]. We derived the radio speed-independent power $P^{cm,ind} = 26.5$mW, radio symbol rate $R = 62.5k$ symbols/sec, and $\beta = 2.74 \times 10^{-8}$ based on CC2420 specification [3] and Eq. (3). We note that the CC2420 is not DMS-capable, so as in [18] we assume four modulation levels, $b = \{2, 4, 6, 8\}$ which give four communicate speeds: $d = \{125, 250, 375, 500\}$ kbps (Eq. (1)). The radio energy is calculated using Eq. (3). We assume a light sensor TSL2561 which takes 12ms to get one reading and consume 0.72mW. Each sensor node uses a rechargeable battery with capacity $\Gamma^{max} = 1000$ joules. All nodes start with the same initial energy level, $\Gamma^{init} = 600$ joules.

Nodes operate in either *normal* or *emergency* mode. We represent the emergency mode by increasing the frame-based workload by w times upon the normal mode, where w is a tunable parameter. This reflects the fact that nodes will need to perform additional duties during those times. We simulate emergency scenarios by introducing contaminant into the system at random time. This can be done by deteriorating the water quality at the water reservoir or a residence. As the contaminant spreads out, the water quality in the residences will decrease and finally been detected by sensor nodes. A sensor node then switches to emergency mode and perform additional workloads over a series of epochs, until the water quality returns to normal.

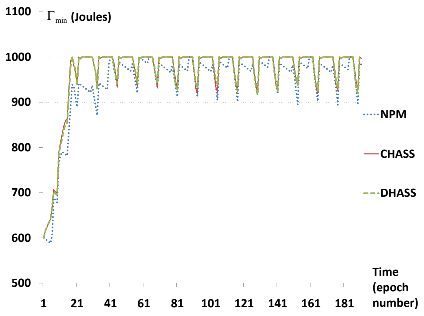
We consider three different types of emergency scenarios. The first type is *random* (RAND) attack. In this case, nodes fail according to a negative exponential distribution, and are picked according to a random uniform distribution from among all the nodes still operating in *normal* mode. The second mode is a *spreading attack* (SPRD). This represents an emergency that increases its area of impact over time. We introduce contaminant into the system from one randomly selected contamination source node. The contaminant spreads out of the system with the flow of water, and lasts a few epochs until water valves are shut off to stop further spreading. The third mode is *area instant* (INST) attack under which a large contiguous area of the network is affected. We simulated one emergency in each horizon, while an emergency started in one horizon may continue to affect multiple successive horizons.

6.2 Results

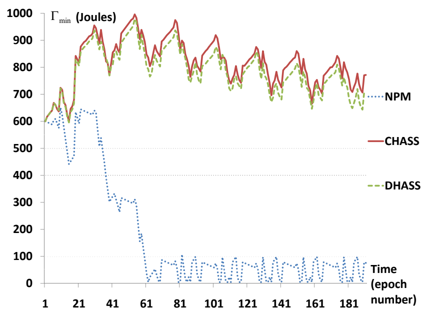
In *CHASS* scheme, the set *EL* contains 2400 elements, given 6 CPU frequencies, 4 modulation levels, and 100 nodes. The experiment setting of *DHASS* is given in [20]. We evaluated the performance of our algorithms under normal and all three emergency modes. We also varied emergency workload levels.

In Fig. 1a-2b, we compared different schemes in term of the achieved Γ_{min}, while assuming non-aggregating applications. We fix the emergency level w at 3.0 which means the emergency workload is three times the normal workload. In normal mode (Fig. 1a), the Γ_{min} value can be seen to vary semi-repetitively, i.e. though it stays close to full capacity at most time, however drops down twice in every horizon (24 epochs). This is because the workload and energy demand in normal mode is relatively low, such that the energy level is dominantly affected by the amount of harvested water energy which varies in repetitive pattern. However in Fig. 1b-2b, the significantly increased workload demand turns to have a dominant effect on energy level, therefore one emergency in each horizon leads to one drop of Γ_{min} in each horizon. This observation demonstrates the effects of harvested energy and workload demand over energy level when operating in different work modes.

As seen from all above figures, in normal and all emergency modes, the *CHASS* scheme achieves the highest Γ_{min}, followed by *DHASS* with slightly lower Γ_{min}. In normal mode (Fig. 1a), *NPM* scheme is able to support Γ_{min} close to full capacity, this is because the harvested energy is much larger than energy demand in normal mode which keeps the energy storage at high level. However, as workload demand increases in emergency mode (Fig. 1b-2b), the performance of *NPM* drops dramatically: its achieved Γ_{min} drops to zero after the 61^{th} epoch in all emergency modes. This implies that at least one node in the network fails to maintain non-empty energy storage and is forced to stop operation. The failure of these nodes will cause service interruption to the entire data collection application during the rest of the epochs. Such lasting service interruption is apparently unacceptable to the mission-critical applications. On the other hand, both *HASS* approaches achieve much higher Γ_{min} than *NPM*. In fact, *CHASS* and *DHASS* never drop to zero in RAND and SPRD modes, and only becomes zero during

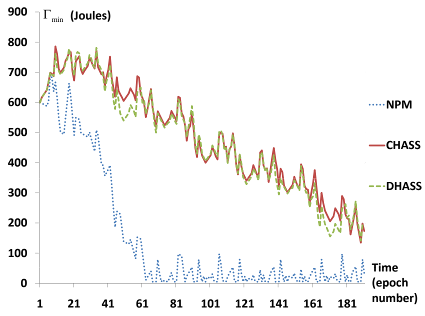


a. Normal mode

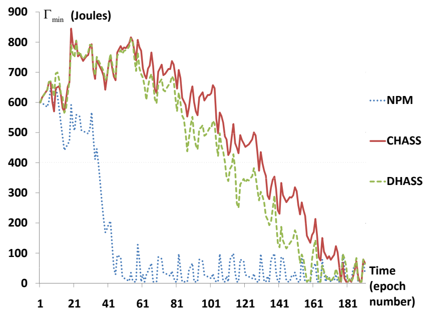


b. RAND mode

Fig. 1. Min. energy level Γ_{min}



a. SPRD mode



b. INST mode

Fig. 2. Min. energy level Γ_{min}

the last ten epochs in INST mode. This is because by using *HASS* approaches, the harvesting-rich nodes run at faster speeds to allow the harvesting-poor nodes to slow down, given tight end-to-end latency constraint. The reduced speeds allow the poor nodes to maintain a higher energy storage level, hence enhance the system's capacity to deal with emergencies. Although under extremely intensive emergency, zero Γ_{min} is inevitable even using the *HASS* approaches, it nevertheless demonstrates the importance of in-network data aggregation with regard to energy efficiency.

Another observation from our results is that *DHASS* closely matches the performance of *CHASS* in term of the value of Γ_{min}. In normal mode (Fig. 1a), the achieved Γ_{min} of *CHASS* and *DHASS* almost overlap. In emergency modes, their performance difference enlarges slightly due to increased influence of workload demands over energy level. This observation indicates that *DHASS* scheme can achieve near-optimal performance. Also, we observed that *DHASS* occasionally achieves higher performance than *CHASS*, e.g. between the 1^{st} and 21^{st} epoch in Fig. 2b, this is due to the energy overheads caused by packet collisions and retransmissions.

Table 1. Percent of depleted nodes: NPM

w	1	1.5	2.0	2.5	3.0
RAND	0%	0%	7%	10%	10%
SPRD	0%	0%	6%	9%	9%
INST	0%	0%	7%	10%	16%

We then conducted a *stress test* over the system while using different schemes. That is, we raise the intensity of emergency by increasing the value of w from 1.5 to 3.0 with an increment of 0.5. The aim of this stress test is to evaluate the resilience of different schemes to various emergency intensities. We measure the system resilience to emergency in term of the percentage of nodes that ran out of energy at the epoch which has the lowest Γ_{min} among all 192 epochs. The smaller the percentage of depleted nodes under the same emergency intensity, the higher resilience supported by a scheme compared to others. Table 1 gives the percentage of depleted nodes in all three emergency modes under various emergency intensities, using *NPM* scheme. As seen from Table 1, as emergency intensity increases, the percentage of depleted nodes increases noticeably in all modes when using *NPM*, which implying the low resilience of the harvesting-unaware NPM scheme to emergency situations. While using both *CHASS* and *DHASS*, the same increase in emergency intensity depletes almost no node in the network, except for the scenario when operating in INST mode with a intensity level $w = 3.0$. The results of the stress test demonstrates the benefit of our harvesting-aware approaches in mitigating the impact of emergencies over the system. We then repeated the same set of experiments above for aggregating applications, the results can be found in [20]. Due to in-network data aggregation, the network traffic pattern and workload demands across nodes are quite different to non-aggregating case, hence it would be very interesting to evaluate our solutions for this type of applications.

7 Conclusion

This paper presented an epoch-based approach for energy management in performance constrained WSNs that utilizing energy harvesting combined with DVS and DMS. We adjust radio modulation levels and CPU frequencies in order to satisfy performance requirement, while maximizing the minimum energy reserve over any node in the network. Through this objective, we ensure highly resilient performance under both normal and emergency situations. Through extensive simulations we demonstrated significant performance improvement by using both our solutions over a baseline scheme.

References

1. Gnawali, O., Fonseca, R., Jamieson, K., Moss, D., Levis, P.: Collection tree protocol. In: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, SenSys 2009, Berkeley, California, November 4-6, pp. 1–14. ACM, New York (2009)
2. Aydin, H., Melhem, R., Mosse, D., Alvarez, P.M.: Power-aware Scheduling for Periodic Real-time Tasks. IEEE Transactions on Computers 53(5), 584–600 (2004)
3. Texas Instrument. CC2420 Datasheet, `http://docs.tinyos.net/index.php/CC2420`
4. Marvell Technology, Xscale PXA27x Datasheet, `http://www.intel.com/design/intelxscale`
5. Gu, Y., Zhu, T., He, T.: ESC: Energy Synchronized Communication in Sustainable Sensor Networks. In: The 17th International Conference on Network Protocols, Princeton, NJ (October 2009)

6. Kansal, A., Hsu, J., Zahedi, S., Srivastava, M.B.: Power management in energy harvesting sensor networks. ACM Trans. Embed. Comput. Syst. 6(4), 32 (2007)
7. Kumar, G.S.A., Manimaran, G., Wang, Z.: End-to-End Energy Management in Networked Real-Time Embedded Systems. IEEE Transactions on Parallel and Distributed Systems, 1498–1510 (November 2008)
8. Liu, S., Qiu, Q., Wu, Q.: Energy aware dynamic voltage and frequency selection for real-time systems with energy harvesting. In: Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2008, Munich, Germany, March 10-14, pp. 236–241. ACM, New York (2008)
9. EPANET 2.0. Water supply and water resources. US EPA (2010)
10. Moser, C., Thiele, L., Benini, L., Brunelli, D.: Real-Time Scheduling with Regenerative Energy. In: Proceedings of the 18th Euromicro Conference on Real-Time Systems, ECRTS, July 5-7, pp. 261–270. IEEE Computer Society, Washington (2006)
11. Moser, C., Thiele, L., Brunelli, D., Benini, L.: Robust and low complexity rate control for solar powered sensors. In: Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2008, Munich, Germany, March 10-14, pp. 230–235. ACM, New York (2008)
12. Noh, D.K., Wang, L., Yang, Y., Le, H.K., Abdelzaher, T.: Minimum Variance Energy Allocation for a Solar-Powered Sensor System. In: Krishnamachari, B., Suri, S., Heinzelman, W., Mitra, U. (eds.) DCOSS 2009. LNCS, vol. 5516, pp. 44–57. Springer, Heidelberg (2009)
13. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: TAG: a Tiny AGgregation service for ad-hoc sensor networks. SIGOPS Oper. Syst. Rev. 36, 131–146 (2002)
14. Shah, P., Shaikh, T.H., Ghan, K.P., Shilaskar, S.N.: Power Management Using ZigBee Wireless Sensor Network. In: Proceedings of the 2008 First International Conference on Emerging Trends in Engineering and Technology, ICETET, July 16-18, pp. 242–245. IEEE Computer Society, Washington (2008)
15. Pitchford, et al.: Inventors, Systems and methods for generating power through the flow of water, US Patent 7,605,485 (issued October 20, 2009)
16. Crossbow Technology. iMote2 Datasheet, http://docs.tinyos.net/index.php/Imote2
17. Yeh, C., Fan, Z., Gao, R.X.: Energy-aware data acquisition in wireless sensor networks. In: IEEE Instrumentation and Measurement Technology Conference (2007)
18. Yu, Y., Krishnamachari, B., Prasanna, V.: Energy-latency tradeoffs for data gathering in wireless sensor networks. In: IEEE Infocom (2004)
19. Zamora, N.H., Kao, J., Marculescu, R.: Distributed power-management techniques for wireless network video systems. In: Proceedings of the Conference on Design, Automation and Test in Europe, Design, Automation, and Test in Europe. EDA Consortium, San Jose, CA, Nice, France, April 16-20, pp. 564–569 (2007)
20. Zhang, B., Simon, R., Aydin, H.: Energy management for time-critical energy harvesting wireless sensor networks, http://cs.gmu.edu/~simon/tr-2010-ehwsn.pdf
21. Zhang, B., Simon, R., Aydin, H.: Joint Voltage and Modulation Scaling for Energy Harvesting Sensor Networks. In: International Workshop on Energy Aware Design and Analysis of Cyber Physical Systems (WEA-CPS), Stockholm, Sweden (April 2010), http://cs.gmu.edu/~simon/weacps10.pdf