Department of Computer Science
George Mason University
Technical Report Series

4400 University Drive MS#4A5
Fairfax, VA 22030-4444 USA
http://cs.gmu.edu/
703-993-1530

Energy Management for Time-Critical Energy Harvesting Wireless Sensor Networks

Bo Zhang bzhang3@gmu.edu
Robert Simon simon@cs.gmu.edu
Hakan Aydin aydin@cs.gmu.edu

Technical Report Aug 2010

Abstract

As Cyber-Physical Systems (CPSs) evolve they will be increasingly relied to support time-critical monitoring and control activities. Further, many CPSs that utilizing Wireless Sensor Networking (WSN) technologies will require the use of energy harvesting methods to extend their lifetimes. For this application class, there are currently few effective models that allow the simulation and analysis of new algorithms or system performance. To address this problem, we define a general purpose WSN model to support a time-critical CPS system. We then present a set of Harvesting Aware Speed Selection (*HASS*) algorithms. Our technique maximizes the minimum energy reserve for all the nodes in the network, thus ensuring highly resilient performance under emergency or fault-driven situations. We present an optimal centralized solution, along with an efficient, fully distributed solution. We propose a CPS-specific experimental methodology, enabling us to evaluate our approach. Our experiments show that our algorithms yield significantly higher energy reserves than baseline methods.

1 Introduction

There is an increasing need to effectively support WSN applications that have significant data collection and processing requirements. Examples range from Wireless Network Video Systems for surveillance [22] to Cyber-Physical Systems such as smart power grid using 802.15.4/Zigbee technology [18] or networks consisting of lab-on-chip nodes [8] used for monitoring large scale water distribution systems. 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 by performing coordinated tracking and flow shut-off operations. Further, many of these 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 [9]. By harvesting energy from environmental sources such as solar, wind or water flow, WSN nodes potentially have a perpetual energy supply. However, given the large energy demands of computational and communication intensive WSN applications, and the potentially limited availability of harvested 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, or the system is experiencing unexpected shortage in environmental energy supply. The focus of this paper is a coordinated energy management policy for time-critical WSN applications that use energy harvesting and that is capable of maintaining required performance level under emergency or fault-driven situations.

Our approach is to make combined use of two energy saving techniques, Dynamic Voltage Scaling (DVS) [3, 16] and Dynamic Modulation Scaling (DMS) [21]. 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. 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 additional difficulty in managing energy for these systems is that nodes may have quite different workload requirements 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 traffics among nodes. 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 levels.

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 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 data processing, communication and performance requirements. Unlike most existing works which assuming solar energy as environmental sources in their simulations, we propose experimental methodology to simulate an energy harvesting WSN systems utilizing energy harvested from water flow in a water distribution system. Our results show that both the centralized and distributed solutions significantly improve the capacity of time-critical WSN systems to deal with emergency situations, in addition to meeting performance requirements. Finally through numerical and experimental analysis, we show that both *HASS* solutions are very efficient with low computational and communication overheads.

2 Background and Related Work

The joint use of DVS and DMS in wireless embedded systems has been explored in [10], [20]. In [10], 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 [20], the energy management problem is formulated as a convex optimization problem, which is then addressed through the use of genetic algorithms. In [21], 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. Unlike our work, [10], [20], [21] assume battery-powered systems without energy harvesting capability. Further, their goals are to prolong system lifetime by reducing energy consumption, without considering issues such as ensuring perpetual operation through energy harvesting or providing for emergency response.

Many existing studies explored the design of energy harvesting WSNs. In [13], 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. ([11]) 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 [15] and [7] which aim at balancing energy supply and demand in energy harvesting WSN systems, and [6], which presents a multi-hop rate assignment approach for data collection. Finally, [9, 14] proposed to maximally utilize harvested energy so as to maximize the amount of completed works, and hence system performance. Neither of these works considered maximizing minimum energy levels 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 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 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 [9]. 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 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 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 [21]. To understand this relationship, let R be the fixed symbol rate. Then modulation level b is associated with communicate speed d expressed as:

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

Let e^{sen} represents the energy required for each sensing which 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 $V_{dd,k}$ [3]. The communication energy e_k^{cm} in the k^{th} frame is a function of communicate speed d_k [21]. Then we have:

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

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

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 constant. The β in Eq. (3) is a constant determined by the transmission quality and noise level [21]. 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. Each 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 [10] [17] [21]. 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 or-

der of *sense→compute→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 [10], [20], [21]. 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 $L_{tot,k}$ of 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 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:

$$Max. \quad \Gamma_{min} \tag{10}$$

$$s.t. \quad \forall \rho_i, H_i \leq \pi \tag{11}$$

$$\forall V_i, f_i \in [f_{min}, f_{max}] \tag{12}$$

$$\forall V_i, d_i \in [d_{min}, d_{max}] \tag{13}$$

$$\forall V_i, 0 < \Gamma_i \leq \Gamma^{max} \tag{14}$$

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 (14) requires that the energy level of any node V_i must be confined to the range $(0, \Gamma^{max}]$. In [9], 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 (14) (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 constant 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 (14). This enables us to remove constraint (14) 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 the Appendix. 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

We call the centralized *HASS* algorithm as *CHASS*, given 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 in which FS_z is the first problem encountered with positive answer, $z = Min.\{p \in [1, |EL|] | ans_p = TRUE\}$ (line (4)). 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; they 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 ρ_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** $[ans, F^*, D^*]$

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 the speed configuration of V_i, $(F^*[i], D^*[i])$ which satisfies:

$$F^*[i] \in [f_{min}, f_{max}], D^*[i] \in [d_{min}, d_{max}] \quad (15)$$

$$\Gamma_i(F^*[i], D^*[i]) \geq EL[p] \quad (16)$$

$$\forall (f, d), l_i(F^*[i], D^*[i]) \leq l_i(f, d) \quad (17)$$

$\Gamma_i(f, d)$ 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. (17). Then 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, hence 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}.

It is possible to reduce the runtime of the above algorithm. In order to do so we present Lemma 2 and Corollary 1, which is used as the basis for $CHASS^*$, the faster algorithm. The key idea of $CHASS^*$ is to implement a binary search for FS_z. This reduces the number of iterations in *CHASS* from $O(|EL|)$ to $O(\log(|EL|))$.

Lemma 2 *For any node V_i, the least per-node latency found by invoking algorithm* Is-Feasible *with Γ_1 as input is no smaller than the one found with Γ_2 as input, where $\Gamma_1 \geq \Gamma_2$.*

The proof of Lemma 2 can be found in the Appendix. Given Lemma 2, we have:

Corollary 1 *For any node V_i, the least per-node latency found in iteration p is no smaller than the one found in iteration q, $\forall q \geq p$.*

Corollary 1 holds because $EL[p] \geq EL[q]$, given that EL was sorted in non-increasing order. Corollary 1 implies that the latency of any path found in iteration p is also no smaller than the one found in iteration q, $\forall q \geq p$. Therefore, we can implement the search for FS_z using binary search. The search starts from the p^{th} element of EL, $p = \frac{|EL|}{2}$, and

- continues on the left half (i.e. $[1, p-1]$) if FS_p has positive answer. Due to smaller path latency found in iteration q, $q > p$, any FS problem on the right half must also have positive answer, hence it is unnecessary to search that half. Rather, on the left half, we may find a problem FS with larger achievable Γ_{min}.
- continue on the right half (i.e. $[p+1, |EL|]$) if FS_p has negative answer. Due to even larger path latency found in iteration q where $q < p$, any FS on the left half must violate constraint (11), thus have negative answer.

The binary search continues on either half depending on the answer to FS_p, until FS_z is found. The binary search based implementation reduces the number of iterations in *CHASS* from $O(|EL|)$ to $O(\log(|EL|))$.

Complexity analysis: Given mn speed configurations, the run time of *find_fastest* is $O(mn)$. Given N nodes, the loop in line (2-4) of Algorithm 2 has run time $O(mnN)$. Also, computing the latency for all paths (line (5)) takes $O(N)$ time since there are N nodes. Therefore, the run time of *Is-Feasible* is $O(mnN)$. Since $CHASS^*$ iterates for $O(\log(|EL|))$=$O(\log(mnN))$ rounds, its total complexity is $O(mnN\log(mnN))$. Since m and n are typically much smaller than N, the algorithm can be seen as an efficient one in practice. In terms of the communication overhead, the gathering of initial energy levels from all the nodes at BS requires one round of data collection.

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*. In each computation round, the midpoint of the search space is used as input energy level to *Is-Feasible*. Given that input, each node calls function *find_fastest* individually to derive its fastest speed configuration and associated per-node latency. It then computes the accumulative latency at it, i.e. L_i+l_i and sends to its parent as a *latency* message. The parent computes its accumulative latency as well based on the received latency messages from its children. By making all the nodes compute and report their latencies accumulatively, the latency of the entire tree L_{tot} will be ultimately computed at the root. The root then compares L_{tot} to π in order to determine the answer to problem FS, and disseminates it to all the nodes as a *decision* message. Note that any node in the network can be the root. The specification of *DHASS* is given below.

In the initialization round, each node estimates its local Γ^{low}, Γ^{high} using Eq. (8), then forwards to its parent as an initialization message. A node receives initialization messages from its children, and compares Γ^{low}, Γ^{high} received to the ones of its own, in order to derive Γ^{low} and Γ^{high} among its children and itself. The derived Γ^{low}, Γ^{high} are sent to its parent as well. When the root receives initialization messages from all its children, it derives the global Γ^{low} and Γ^{high} which actually equals to the minimum and maximum elements in EL, $Min(EL)$, $Max(EL)$. Then the root disseminates the global Γ^{low} and Γ^{high} to all the nodes for the use in the first computation round. Their values will be updated in each computation round according to a rule given in the following paragraphs. The initialization round ends when all the nodes receive them.

In a computation round, each node V_i calls function *find_fastest* to derive its fastest speed configuration and least per-node latency l_i^{min}, while satisfying $\Gamma_i \geq X$. X=$(\Gamma^{low}+\Gamma^{high})/2$ is the input energy level in this round. Note that X should be the same for all nodes since they had agreed on the initial Γ^{low} and Γ^{high} after the initialization round, also the updating rule of these two values is the same for all nodes. Then, starting from the leaf nodes, each node sends the accumulative latency at it, i.e. L_i+l_i^{min} to its parent. Recall that L_i=0 for any leaf node. Any non-leaf node computes its L_i using Eq. (6) after received latency messages from all children. The computation and reporting of L_i+l_i^{min} continues on all nodes in the tree. The root node computes the latency of the entire tree L_{tot} after received *latency* messages from all children. Then the root compares L_{tot} to the latency constraint π. If $L_{tot} \leq \pi$, the root sets the answer to problem FS in the current round to be positive; otherwise, it sets the answer to be negative. Unlike the centralized version, we compute and compare (to π) the latency of the entire tree instead of the latency of each individual path. This is because in each computation round, it requires a node sending only the latency of the subtree rooted at it, rather than one for each path it resided on.

The answer to problem *FS* is then disseminated to all the nodes as a *decision* message. Each node receives the *decision* message, and continues to the next round. The search space $[\Gamma^{low}, \Gamma^{high}]$ and input energy level X in the next round are updated based on the answer in the received *decision* message as follows:

- If answer is positive, we set $\Gamma^{low} = X$ which directs the search to the higher-valued half of the search space.
- If answer is negative, we set $\Gamma^{high} = X$ which directs the search to the lower-valued half.
- We then set $X = (\Gamma^{low} + \Gamma^{high})/2$.

The algorithm ends after Y computation rounds, and sets:

- X in the Y^{th} round to be Max. Γ_{min}, and the speed configuration found in the Y^{th} round to be the opti-

mal solution, if the answer to the FS problem in the Y^{th} round is positive.

- Γ^{low} in the Y^{th} round to be the Max. Γ_{min}, and the speed configuration found in the round with $X = \Gamma^{low}$ as the input to be the optimal solution, otherwise.

We then propose Theorem 1 which shows the performance of *DHASS* is very close to optimal.

Theorem 1 *The maximum* Γ_{min} *found by algorithm* DHASS *is smaller than the optimal value, by at most* $(Max(EL) - Min(EL))/2^{Y-1}$.

The proof of Theorem 1 can be found in the Appendix.

Complexity analysis: In a computation round, the major time cost of a node comes from function *find_fastest* which equals $O(mn)$. Given Y computation rounds, the total cost is $O(mnY)$. The root compares L_{tot} to π in each computation rounds, this causes a time cost of $O(Y)$. In the initilization round, each node sends exactly one initialization message. In any computation round, each node sends exactly one latency message. The optimal speed configurations are computed on each node individually, hence there is no dissemination cost.

6 Performance evaluation

We performed a series of simulations to evaluate the effectiveness of our *HASS* approaches. Specifically, the goal of the evaluation is to determine how well both the *CHASS* and *DHASS* algorithm maximize the minimum energy level across the system. We considered two WSN application types: aggregating and non-aggregating application. 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

Without loss of generality 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. Our simulation environment assumes that each water meter is coupled with a DVS-DMS enabled node. Energy is harvested from the flow of water. 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 energy source.

We have developed simulation software upon TOSSIM: the standard high-fidelity WSN simulator,

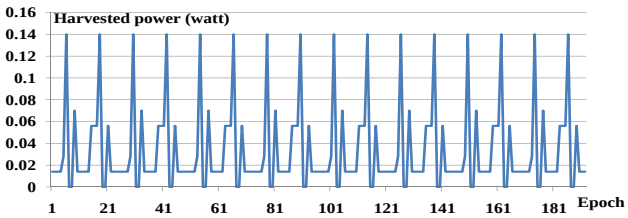

Figure 1: Water energy harvesting profile

combined with EPANET [12]: 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 houses then form a WSN system. We use *Collection Tree Protocol* [1] to organize the nodes into a data collection tree.

Due to the repetitive water usage pattern with a cycle of 6 hours as supported by 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 which is a sufficiently long duration, 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. Fig. 1 shows 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 on 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 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; the computational workload is randomly selected between C=$[0, 3000000]$ cy-

cles. These two scenarios produce highly different levels of workload and network traffic.

The hardware basis for a DVS-DMS capable platform is the widely available iMote-2 sensor node [19]. The iMote-2 platform has a Intel Xscale PXA27x CPU [5] and a ChipCon CC2420 radio [4]. The frequency and power specification of PXA27x processor is given in Table (I). 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 [4] and Eq. (3), and model a DMS-capable radio by assuming four modulation levels: $b = \{2, 4, 6, 8\}$ [21] which gives 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 [2] 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.

$Freq.(MHz)$	104	208	312	416	520	624
$Power(mW)$	116	279	390	570	747	925

Table 1: Specification of Intel Xscale Pxa27x

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 certain residences in EPANET (imagine a terrorist attack on a water supply). As the spreading of contaminant, 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 which affecting the system in different patterns. 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

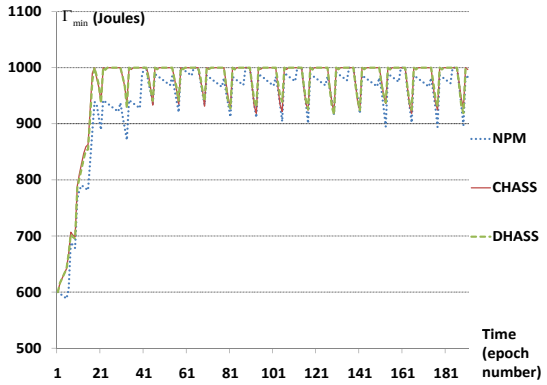

Figure 2: Min. energy level - Normal, non-aggregation

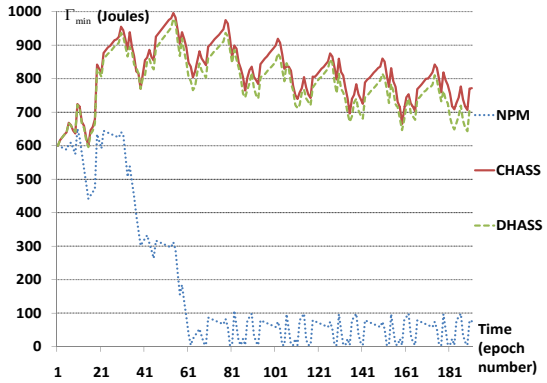

Figure 3: Min. energy level - RAND, non-aggregation

mode is *area instant* (INST) attack. Under this scenario a large contiguous area of the network is affected. We prevent spreading of contaminant by shutting off the water valves. 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. In *DHASS* scheme, we set the number of search iterations to be $\log(|EL|) \approx 11$.

We evaluated the performance of our algorithms under normal and all three emergency modes. Each scenario was tested using complete aggregation and non-aggregation. We also varied emergency workload levels.

6.2.1 Non-aggregating applications

In Fig. 2-5, 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. 2), 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 (as seen in Fig. 1). However in Fig. 3-5, 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. 2), *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. 3-5), the performance of *NPM* drops dramatically: its achieved Γ_{min} drops to zero after the 61^{th} epoch in all emergency modes. This imply 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 side, both *HASS* approaches achieves much higher Γ_{min} than *NPM* which never drops to zero in RAND and SPRD modes, and becomes zero only during 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-weak nodes to slow down, given tight end-to-end latency constraint. The reduced speeds allow the weak 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* achieves a very close Γ_{min} to *CHASS*. In normal mode (Fig. 2), the achieved Γ_{min} of *CHASS* and *DHASS* al-

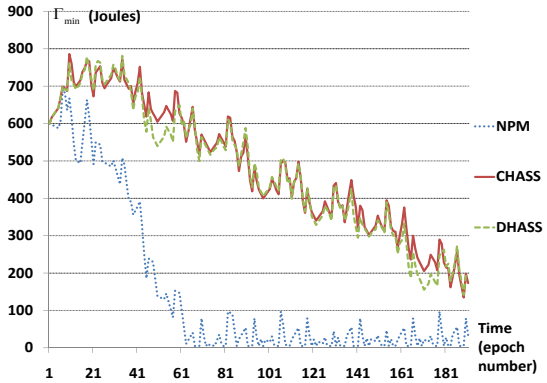

Figure 4: Min. energy level - SPRD, non-aggregation

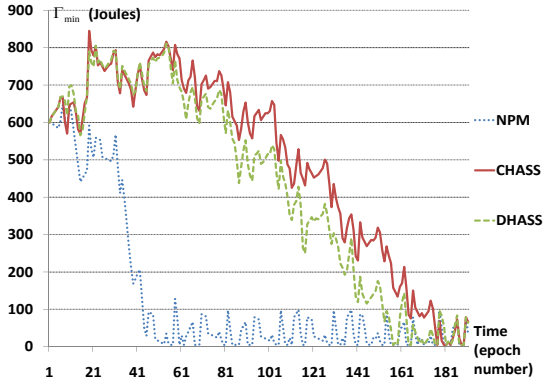

Figure 5: Min. energy level - INST, non-aggregation

most overlap. In emergency modes, the performance difference between them enlarges slightly due to the increased influence of workload demands over energy level. This observation indicates that *DHASS* scheme can achieve near-optimal performance when it runs enough number of binary search iterations, as claimed in Theorem 1. Also, we observed that *DHASS* occasionally achieves higher performance than *CHASS*, e.g. between the 1^{st} and 21^{st} epoch in Fig. 5, this is due to the energy overheads caused by packet collisions and retransmissions.

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%

Table 2: Percent of depleted nodes: NPM, Non-aggregation

Then, we conducted a *stress test* over the system while

w	1	1.5	2.0	2.5	3.0
$RAND$	0%	0%	0%	0%	0%
$SPRD$	0%	0%	0%	0%	0%
$INST$	0%	0%	0%	0%	1%

Table 3: Percent of depleted nodes: CHASS, Non-aggregation

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 2 and 3 give the percentage of depleted nodes in all three emergency modes under various emergency intensities, using *NPM* and *CHASS* scheme respectively. We omit the results using *DHASS*, since it results in exactly the same percentages as *CHASS* under all scenarios. As seen from Table 2, 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 *CHASS*, 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 $w = 3.0$. In that scenario, the lowest Γ_{min} which equals zero, appears around the 181^{st} epoch using all three schemes (as seen in Fig. 5). We then collected the energy levels of all the nodes at that time, and found 16% of depleted nodes when using *NPM*, while only 1% when using both *HASS* schemes. The results of the stress test demonstrates the benefit of our harvesting-aware approaches in mitigating the impact of emergencies over the system.

6.2.2 Aggregating applications

For aggregating applications, we repeat the same set of experiments as for non-aggregating applications. In Fig. 6-9, we also fix the emergency intensity at $w = 3.0$ and plotted the Γ_{min} achieved by using different schemes. In all the modes, *CHASS* and *DHASS* schemes again achieve much higher Γ_{min} than *NPM* scheme. The *DHASS* scheme achieves very close performance to *CHASS* scheme. As seen in all figures, the achieved Γ_{min} by our *HASS* approaches never drop to zero, while the achieved Γ_{min} by *NPM* schemes drops to zero for many times due to the impact of emergency.

Finally, Table 4 and 5 show the percentage of de-

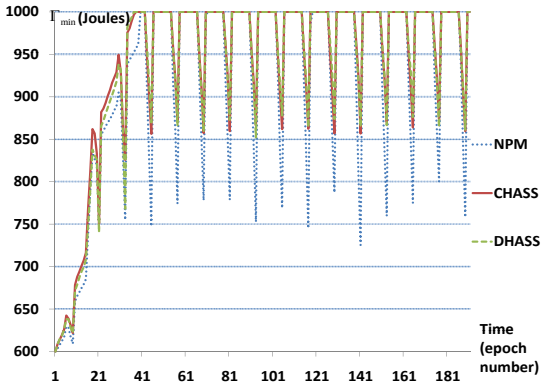

Figure 6: Min. energy level - Normal, complete aggregation

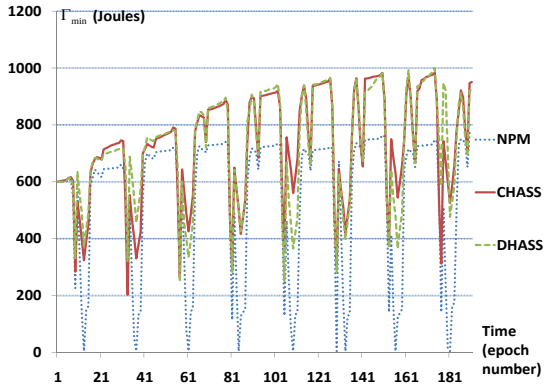

Figure 7: Min. energy level - RAND, complete aggregation

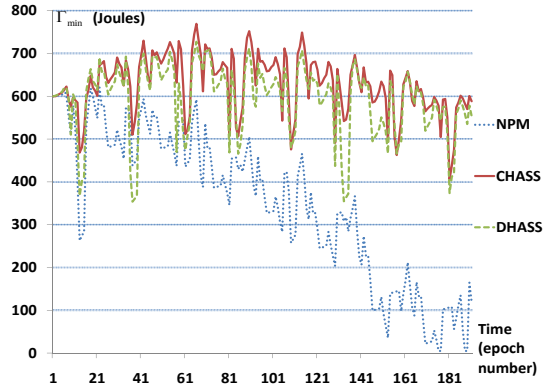

Figure 8: Min. energy level - SPRD, complete aggregation

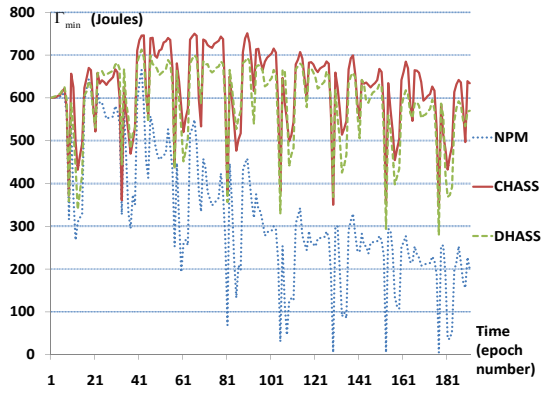

Figure 9: Min. energy level - INST, complete aggregation

w	1	1.5	2.0	2.5	3.0
$RAND$	0%	0%	0%	0%	3%
$SPRD$	0%	0%	0%	0%	18%
$INST$	0%	0%	0%	0%	13%

Table 4: Percent of depleted nodes: NPM, complete-aggregation

pleted nodes under various emergency intensities by using different schemes. As seen from the tables, fewer nodes are depleted in aggregating applications than in non-aggregating applications in all scenarios. Even for *NPM* scheme, nodes are depleted only when the emergency intensity increases to $w = 3.0$. While using both *HASS* approaches, depletion of nodes never happens.

w	1	1.5	2.0	2.5	3.0
$RAND$	0%	0%	0%	0%	0%
$SPRD$	0%	0%	0%	0%	0%
$INST$	0%	0%	0%	0%	0%

Table 5: Percent of depleted nodes: CHASS, complete-aggregation

Communication overhead of algorithm *DHASS*: In the simulation, *DHASS* requires each node to send 1 initialization message, and 11 latency messages for the 11 computation rounds. Since *DHASS* needs to run only once in each epoch, each node sends in total 12 control messages in an epoch. Given a frame period $\pi = 240$ ms, and epoch length $S = 15$ minutes, each node executes 3750 frames and sends 3750 data messages. Therefore we derive the communication overhead to be $12/(3750 + 12) \approx 0.32\%$. Consider that control messages commonly have smaller size than data messages, the energy overhead might be even smaller. This implies that our *HASS* approaches have also the benefit of high efficiency.

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. The goal of our approach is to maximize the minimum energy reserve over any node in the network. Through this objective, we ensure highly resilient performance under both normal and emergency situations. We formulate our problem as an optimization program, and then solve it with centralized and distributed algorithms. Through simulation we have extensively evaluated both algorithms against a baseline scheme. Our results show that our algorithms achieve significantly higher performance than a baseline approach, under both normal and emergency situations.

A Appendix

Proof of Lemma 1:

We denote the optimal solution of problem *HASS-N* as S^N, and the achieved minimum energy level using S^N as Γ^N_{min}. Note that S^N is optimal in the sense that it maximizes Γ_{min}, without considering constraint (14). Similarly, we denote the optimal solution of problem *HASS* as S^*, and its achieved minimum energy level as Γ^*_{min}. We will show that, if S^N satisfies the positivity constraint, i.e. $\Gamma^N_{min} > 0$, then we have $S^* = S^N$; otherwise, S^* does not exist. We consider the following three cases:

- If using S^N, $0 < \Gamma_i \leq \Gamma^{max}$ holds for any node V_i, obviously we have $S^* = S^N$.
- In case that S^N leads to $\Gamma_i > \Gamma^{max}$, $\exists V_i$, we claim that the compute and communicate speeds contained in S^N can still be used. This is because as soon as the maximum capacity of the energy storage is reached, the harvesting circuitry can be automatically turned off, keeping its energy level at Γ^{max}. In another words, in this case, we still have $S^* = S^N$.
- If S^N leads to $\Gamma_i \leq 0$, $\exists V_i$, this implies $\Gamma^N_{min} \leq 0$. Assume S^* exists, then using S^* will lead to $\forall V_i$, $\Gamma_i > 0$ in S^*, since S^* must satisfy the positivity constraint by definition. This indicates $\Gamma^*_{min} > 0 \geq \Gamma^N_{min}$ which contradicts the fact that S^N maximizes Γ_{min} (recall that the feasible region of *HASS* is contained in that of *HASS-N*). Therefore, if S^N violates the positivity constraint, S^* cannot exist.

Proof of Lemma 2:

Let (f_1, d_1) and (f_2, d_2) denote the fastest speed configurations at V_i found when the algorithm FS is invoked by using Γ_1 and Γ_2 as input respectively, where $\Gamma_1 \geq \Gamma_2$. We use $l_{i,1}^{min}$ and $l_{i,2}^{min}$ to denote the (least) per-node latencies at V_i obtained by using (f_1, d_1) and (f_2, d_2), respectively. We will show $l_{i,1}^{min} \geq l_{i,2}^{min}$.

When the FS problem is solved with Γ_1 as input, (f_1, d_1) yields $l_{i,1}^{min}$, while satisfying $\Gamma_i(f_1, d_1) \geq \Gamma_1$. When it is solved with Γ_2 as input, the function *find_fastest* could at least find $(f_2, d_2) = (f_1, d_1)$ which yields $l_{i,2}^{min} = l_{i,1}^{min}$, while satisfying $\Gamma_i(f_2, d_2) \geq \Gamma_2$ (because $\Gamma_i(f_2, d_2) = \Gamma_i(f_1, d_1) \geq \Gamma_1 \geq \Gamma_2$). In many cases, (f_2, d_2) will yield an even smaller $l_{i,2}^{min}$.

Proof of Theorem 1:

Denote Γ^{low} and Γ^{high} in the Y^{th} round as $\Gamma^{low,Y}$ and $\Gamma^{high,Y}$, respectively. According to the property of binary search, in the Y^{th} round, the maximum Γ_{min} which is our search target is confined to the range $[\Gamma^{low,Y}, \Gamma^{high,Y}]$. As a result, the maximum Γ_{min} can be larger than the Γ_{min} found in the Y^{th} round by at most $\Gamma^{high,Y} - \Gamma^{low,Y}$. In the Y^{th} round, $\Gamma^{high,Y} - \Gamma^{low,Y} = (Max(EL) - Min(EL))/2^{Y-1}$.

References

[1] Collection tree protocol. `www.tinyos.net/tinyos-2.x/doc/html/tep123.html`.

[2] Taos tsl2561 datasheet. `http://www.taosinc.com/`.

[3] Hakan Aydin, Rami Melhem, Daniel Mosse, and Pedro Meja-alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Transactions on Computers*, 53:584–600, 2004.

[4] Chipcon. Cc2420. `docs.tinyos.net/index.php/CC2420`, 2006.

[5] Intel Corp. Xscale pxa27x. `www.intel.com/design/intelxscale`.

[6] Kai-Wei Fan, Zizhan Zheng, and Prasun Sinha. Steady and fair rate allocation for rechargeable sensors in perpetual sensor networks. In *SenSys*, 2008.

[7] Y. Gu, T. Zhu, and T. He. Energy synchroinized communication in sustainable sensor networks. *ICNP*, 2009.

[8] W. L. Hwang, Fei S., and K. Chakrabarty. Automated design of pin-constrained digital microfluidic arrays for lab-on-a-chip applications. In *DAC*, 2006.

[9] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava. Power management in energy harvesting sensor networks. *Trans. on Embedded Computing Sys.*, 2007.

[10] G. Sudha Anil Kumar, Govindarasu Manimaran, and Zhengdao Wang. End-to-end energy management in networked real-time embedded systems. *IEEE Trans. Parallel Distrib. Syst.*, 19(11):1498–1510, 2008.

[11] S. Liu, Q. Qiu, and Q. Wu. Energy aware dynamic voltage and frequency selection for real-time systems with energy harvesting. In *DATE*, 2008.

[12] EPANET 2.0. Water supply and water resources. *US Environmental Protection Agency*, 2010.

[13] Benini L. Brunelli D. Moser C., Thiele L. Real-time scheduling with regenerative energy. In *ECRTS*, 2006.

[14] Brunelli D. Benini L. Moser C., Thiele L. Robust and low complexity rate control for solar powered sensors. DATE 08.

[15] Dong Kun Noh, Lili Wang, Yong Yang, Hieu Khac Le, and Tarek Abdelzaher. Minimum variance energy allocation for a solar-powered sensor system. In *DCOSS*, 2009.

[16] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *SOSP*, 2001.

[17] J. Hellerstein W. Hong S. Madden, M. J. Franklin. Tag: a tiny aggregation service for ad-hoc sensor networks. In *OSDI*, 2002.

[18] Shaikh T. Ghan K. Shilaskar S. Shah, P. Power management using zigbee wireless sensor network. In *ICETET*, 2008.

[19] Crossbow Technology. Imote2 datasheet. `www.xbow.com`.

[20] C. Yeh, Z. Fan, and R.X. Gao. Energy-aware data acquisition in wireless sensor networks. In *IMTC. IEEE*, 2007.

[21] Y. Yu, B. Krishnamachari, and V. Prasanna. Energy-latency tradeoffs for data gathering in wireless sensor networks. In *Infocom*, 2004.

[22] N. H. Zamora, J. Kao, and R. Marculescu. Distributed power-management techniques for wireless network video systems. In *DATE*, 2007.