Harvesting-Aware Energy Management for Time-Critical Wireless Sensor Networks with Joint Voltage and Modulation Scaling

Bo Zhang, Robert Simon, *Member, IEEE,* and Hakan Aydin, *Member, IEEE*

***Abstract*—As Cyber-Physical-Systems (CPSs) evolve they will be increasingly relied on to support time-critical and performance-intensive monitoring and control activities. Further, many CPSs that utilize 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 algorithmic techniques that combine performance-sensitive processing and communication with efficient management techniques for energy harvesting. Our paper addresses this problem. We first propose a general purpose, multi-hop WSN architecture capable of supporting time-critical CPS systems using energy harvesting. 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, 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.**

***Index Terms*—Wireless sensor networks, Energy harvesting, Dynamic voltage scaling, Dynamic modulation scaling**

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 [29] to Cyber-Physical Systems such as smart power grid using 802.15.4/Zigbee technology [21] or networks consisting of lab-on-chip nodes [12] used for monitoring large scale water distribution systems. These types of systems are *time-critical* and *performance sensitive*. 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 performance and system availability requirements with energy management policies that can maximize system lifetime.

One approach to maximize the system's lifetime is based on energy harvesting [13] [3]. By harvesting energy from environmental sources such as solar, wind or water flow, WSN nodes may 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 the harvested environmental power, the perpetual operation of WSN nodes cannot be realized without deliberate energy management and performance control. 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 utilizes two energy management techniques, Dynamic Voltage Scaling (DVS) [6] and Dynamic Modulation Scaling (DMS) [28]. 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. 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* framework is to maximize energy reserves while meeting application performance requirements, therefore maximizing the system's resilience in the face of emergency situations. One 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 the differences in nodes' energy harvesting opportunities, unbalanced distribution of processing workloads, or network traffic among nodes. Because of these conflicting design considerations, the *HASS* approach attempts to maximize the energy reserve levels of nodes in the network while guaranteeing the required system performance levels. Our ultimate goal is to maximize system resilience to network-wide workload burst or shortage in the amount of energy harvested.

Our specific contributions are summarized as follows: We first provide a basic architectural description for DVS-DMS capable nodes that use energy harvesting, and a general network and performance model for time-critical WSN applications. Based on our device and application models, we formulate the problem of maximizing the minimum energy reserve while maintaining required performance as an opti-

Bo Zhang, Robert Simon, and Hakan Aydin are with the Department of Computer Science, George Mason University, Fairfax, VA, 22030 USA e-mail: (bzhang3@gmu.edu; simon@cs.gmu.edu, aydin@cs.gmu.edu).

mization problem. We then solve this problem with an optimal and efficient centralized algorithm, along with a distributed version. As a further improvement, we propose a *speed reduction scheme* that increases energy levels by reducing the performance levels of certain nodes while maintaining all energy and system-wide performance constraints. We conducted extensive simulations to evaluate our *HASS* solutions under a variety of data processing, communication and performance requirements. We propose an experimental methodology to simulate a WSN system 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.

2. Background and Related Work

There is currently much interest in developing and deploying time-critical and processing intensive WSN systems [12], [21], [29]. A major motivating factor is that WSN technology is a key enabler of many types of proposed Cyber Physical Systems (CPS) [25]. Many researchers have identified the fundamental requirements of CPS as the need to operate in an unattended and perpetual fashion, with short unavailability periods [27]. These requirements strongly suggest the need to combine methods for energy-harvesting, performance management and emergency response planning. We now briefly review related work in this area.

DVS and DMS based scheduling for wireless embedded systems has been explored in [22], [28]. In [22], 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 [28], Yu et al. proposed a DMS-based approach for tree-based WSNs which periodically collect data from sensor nodes. Their objective is minimizing the network-wide energy consumption. Unlike our work, [22], [28] assume battery-powered systems without energy harvesting capability.

Many existing studies explored the use of DVS technique for energy harvesting WSNs. In [19], 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. [14] proposed EA-DVFS (Energy-Aware Dynamic Voltage and Frequency Scaling) which improves energy efficiency of LSA by using DVS. [34] proposes an improved version of EA-DVFS which further enhances energy efficiency and reduces deadline miss ratio. Both LSA and EA-DVFS manage only the CPU energy and ignore the radio energy. [33] proposes a novel task scheduler based on the linear regression model. The scheduler aims to improve the task accuracy and the number of measurements taken. DVS is used to improve energy usage efficiency. Additionally, [11], [30], [31] [1], [4] studied the design of real-time wireless sensor networks. [11] proposed energy aware routing approach for real-time and reliable communication in industrial WSN system. [30] uses two-hop velocity information to enhance real-time data delivery in WSNs. [31] uses a topology management approach to ensure real-time and energy-efficient data communication. [1] proposes a scheduling algorithm for minimizing energy consumption of nodes inside a cluster tree while meeting all the deadlines of pre-defined data flows. [4] proposes a message scheduling algorithm for guaranteeing real-time communication between sensor nodes by controlling parameters of 802.15.4 MAC, such as beacon order, super-frame order. Further, [2] targets prolonging the lifetime of sensor nodes by dynamically adjusting their sleep schedules, while matching the network demands.

Many other works studied the design of energy harvesting WSNs. For instance, the work in [17] presents a multi-parametric set of algorithms for predicting harvesting rates and setting service levels. Similar to our approach, this work uses a task-oriented control method. Additional research including [9], [13], [18], [35] propose to maximally utilize the harvested energy so as to maximize the amount of completed work, and hence the system performance. System performance was correlated to nodes' duty cycles in [13], data sampling rate in [9], [18], and application-defined utility values in [35]. None of these papers considers maximizing the minimum energy levels by using joint DVS-DMS techniques. The assumption of these studies is that the more workload the sensor nodes complete, the higher the quality of results of the application. We note that many CPS applications do not have this requirement. Rather, the number of sense tasks and associated computation and communication activities are fixed by the application and criticality of the task. Unlike these papers, our goal is to enhance the system's ability to tolerate unexpected energy usage and energy shortage by maintaining sufficient energy storage, given a fixed application-defined performance requirement.

Part of the motivation for defining our epoch based approach is to provide a complementary node-based solution that can be used within network-wide energy management policies and architectures, such as those presented in [5], [7], [10]. Our work is easily paired with these energy aware architectural approaches, since our optimization formulation can be used as the application-defined reward or utility metric. For instance, the *IDEA* approach [7] provides a network-level service by distributing performance and state information allowing nodes to optimize application-based performance metrics. Our control objectives and methods for determining epoch length and frame type could therefore be implemented within an *IDEA*-like architecture.

3. System Architecture and Assumptions

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.

A. 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 the required sensor suites. The harvester head is energy source-specific, such as a solar panel or wind generator. The energy storage unit (e.g. rechargeable 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 [13]. 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, where i is the epoch number. P_i remains constant within the course of each epoch i, but changes for different epochs. The time unit used for harvesting prediction is therefore one epoch. In our approach, one needs to know the harvested power prediction only for the next epoch.

The node consumes power via either processing, radio communication or sensing. We now describe how to model energy consumption of an individual node. The basic time interval over which energy consumption is calculated is called a *frame*, defined below. Frames are invoked periodically, and each prediction epoch consists of multiple frames.

We assume the DVS-enabled CPU has m discrete frequencies $f_1<...<f_m$ in units of cycles/second, and the DMS-enabled radio has n discrete modulation levels, $b_1<...<b_n$. We use the terms CPU frequency and computation speed interchangeably. In practice, the modulation level represents the number of bits encoded in one signal symbol [28]. Let R be the fixed symbol rate. Then the modulation level b is associated with the communication speed d expressed as:

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

The computation power P^{cp} is a function of computation speed f. Given m discrete frequency levels $\{f_1, ..., f_m\}$, there are m different levels of computation power, $\{P^{cp}(f_1), ..., P^{cp}(f_m)\}$. [6] models P^{cp} as having a cubic relation to f. The communication power P^{cm} is a function of communication speed d. Given n discrete levels of communication speeds d, there are n different radio power levels $\{P^{cm}(d_1), ..., P^{cm}(d_n)\}$. Following [28] P^{cm} is considered to have an exponential relation with d. Let C and M be the computation and communication workloads in a frame, where C is the number of CPU cycles to be processed, and M is the number of bits to be transmitted. The computation and communication energy in a frame, denoted by e^{cp} and e^{cm} are function of f and d respectively, given below:

$$e^{cp}(f) = P^{cp}(f) \cdot (C/f) \tag{2}$$

$$e^{cm}(d) = P^{cm}(d) \cdot (M/d) \tag{3}$$

Since we target time-critical and performance senstitive WSN applications, we assume a sufficient level of coordinated sleeping and time-slotted transmission scheduling, so that the radio energy consumed by listening channel activities is not a significant factor. Let e^{sen} represent the constant energy required for each sensing operation. We can give the total energy consumed in a frame, e^c as:

$$e^c(f,d) = e^{sen} + e^{cp}(f) + e^{cm}(d) \tag{4}$$

B. Network and application model

The system consists of N sensor nodes $V_1, \ldots, V_n$. The base station is denoted by BS. The sensor nodes are divided into two types: *source* nodes perform sensing, processing and

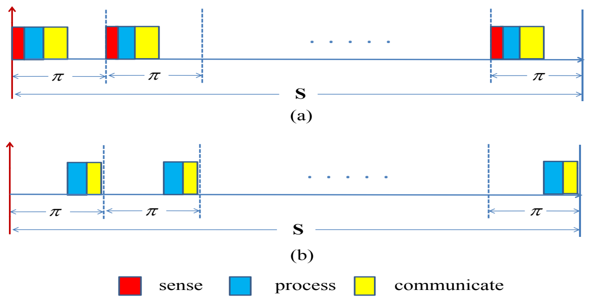


Fig. 1: Frame-based Data Collection. S: epoch length, π: frame length. Figure 1(a) and (b) show a sequence of frames invoked on a source node and a relay node, respectively

communication operations, while *relay* nodes only perform processing and communication. Our data processing architecture is 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 the source nodes report their readings every π time units. The time interval π is the length of each *frame*. Such a frame-based data collection mechanism is quite common for WSN applications [15], [22], [28]. All sensed, processed or aggregated data must reach the 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 their readings 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 the nodes are time-synchronized so that they are aware of the same frame start and end times. Figure 1(a-b) show a sequence of frames executed by the source nodes and relay nodes within an epoch, along with the relationship of frames to a prediction epoch respectively. Given the epoch length S and frame length π, there are $\lfloor \frac{S}{\pi} \rfloor$ frames in any epoch. The source nodes perform sensing, processing, and communication tasks (Fig 1(a)); the relay nodes only process and forward data (Fig 1(b)).

On a per-frame basis, energy consuming activities within each node are represented using a task-based model. In this way, the 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*. We assume in each frame, a sensor node performs sensing first, then process the sensor reading (computation), and finally transmit the processed data (communication). 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. The packet size M_i can be obtained based on the packet format specified by the application, and will be upper bounded by the data link layer, such as 127 bytes in 802.15.4. The CPU cycle count C_i can be estimated using existing techniques such as [32].

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 the computation and communication speeds only change at the start of an epoch. This design decision reduces the required level of control and synchronization overhead.

Using this analysis we can calculate the time required by each node V_i to carry out all the activities during a frame, referred as the *per-node latency*, l_i. The per-node latency depends upon the computation speed f_i and communication speed d_i. Then l_i is given by

$$l_i = t^{sen} + \frac{C_i}{f_i} + \frac{M_i}{d_i} \tag{5}$$

t^{sen} is the sensing time which is a constant. Note that t^{sen} is zero for relay nodes. We assume that the effective data transmission time dominates the overall communication time while ignoring the carrier sense time [22], [28]. Thus, the communication time is inversely proportional to d_i.

The system is organized into a data collection and processing tree rooted at BS. We assume the availability of a real-time and reliable routing protocol for time-critical and performance sensitive WSNs, such as EARQ [11]. In order to support time-critical operation we must define and calculate the *path latency* and *data collection latency*. These two values are used in the optimization formulation in Sections 4 and 5 to ensure that all latency requirements are maintained.

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 of ρ_i is defined as:

$$H_i = \sum_{j:V_j \in \rho_i} l_j \tag{6}$$

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, $Parent(V_i)$ after receiving data from all its children. Then, we define the *collection latency* of the subtree rooted at node V_i, denoted as L_i, in a recursive fashion:

$$L_i = Max.\{L_j + l_j | V_j \in Children(V_i)\} \tag{7}$$

L_i is the latency for V_i to collect data from any source nodes in the subtree . The subtree rooted at a leaf node contains only the leaf itself, and hence incurs zero latency. Then the *total collection latency* of the entire tree L_{tot} is given by

$$L_{tot} = Max.\{L_i + l_i | V_i \in Children(BS)\} \tag{8}$$

L_{tot} is the time interval between the start of a frame, and when BS collects all sensed data. Note that by resolving the recursion in Eq. (8), L_{tot} actually equals to the latency of the longest path in the tree, i.e. $Max.\{H_i | \forall \rho_i\}$.

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 in the network. The computation and communication speeds at individual nodes are adjusted at the start of each epoch, and remain fixed throughout that epoch. As defined in Section 3-A, an epoch is a time interval over which an energy harvesting prediction can be reasonably made. Within an arbitrary epoch, the energy consumption e_i^c, and the latencies L_i, H_i of a node V_i are fixed over any frame k. Then the energy level Γ_i of a node V_i at the end of a given epoch is given as:

$$\Gamma_i = \Gamma_i^{init} + P_i \cdot S - \lfloor S/\pi \rfloor \cdot e_i^c \tag{9}$$

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:

$$\Gamma_{min} = Min.\{\Gamma_i | \forall V_i\} \tag{10}$$

Then the goal of our approach is to maximize Γ_{min}. We achieve this goal by regulating nodes energy level and hence Γ_{min} through adjustment of two variables, the computation and communication speeds f_i, d_i of any node V_i. 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:

$$\begin{aligned} Max. \quad & \Gamma_{min} && (11)\\ s.t. \quad & \forall \rho_i, H_i \le \pi && (12)\\ & \forall V_i, f_i \in \{f_1, ..., f_m\}, d_i \in \{d_1, ..., d_n\} && (13)\\ & \forall V_i, 0 < \Gamma_i \le \Gamma^{max} && (14) \end{aligned}$$

The constraint (12) ensures that the latency of any path ρ_i in the tree is smaller than the frame period π. As mentioned in Section 3-B, this is equivalent to ensuring that the collection latency of the entire tree is smaller than π. The constraint (13) 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 $(0, \Gamma^{max}]$. The left-hand side of the constraint (14) (called the *positivity constraint*) must hold in order to ensure positive energy level at any time, so that an energy harvesting sensor node can operate without any interruption. The right-hand side (called the *capacity constraint*) is used to model the energy storage capacity.

V_i	A node in the network
$CHILDREN(V_i)$	The set of children of node V_i
N	Number of nodes in the network
Γ_i	Energy storage level of node V_i
S	Epoch length
P_i	Harvested power of node V_i
f, d	Computation and communication speed
e^{sen}, e^{cp}, e^{cm}	Unit sensing, computation and communication energy
C, M	Computation and communication workload
π	Data collection frame period
l_i	Per-node latency of node V_i
L_i, L_{tot}	Data collection latency at node V_i and BS
ρ_i	Path from node V_i to base station
H_i	Latency of path ρ_i

TABLE I: List of notations

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 the problem *HASS* with the 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*. 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 computation speeds and n communication speeds, there are $(mn)^N$ possible solutions, making brute force search impractical.

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 (12-13). 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} values is referred as EL (Energy Level), and has at most mnN elements.

A. Centralized version

We call the centralized *HASS* algorithm *CHASS*; this is shown in *Algorithm 1*. It runs on the base station, and requires that BS collects Γ^{init} from each node in the system. It is also aware of the available speed configurations of sensor nodes. *CHASS* first computes the possible energy levels of all the nodes using Eq. (9) 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* (FS_p), by calling the 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 $\Gamma_{min} = EL[p]$, while satisfying the constraints (12-13)?"

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 discovers a FS_p with positive answer. In other words, it terminates during 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, 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 returns three values, the boolean answer to problem FS_p, ans_p, and two speed vectors 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:     $[ans_p, F^*_p, D^*_p]$ = 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:     $(F^*[i], D^*[i], l_i^{min}) = find\_fastest(\Gamma_{min}, V_i)$
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:     $ans = FALSE$, $F^* = \emptyset$, $D^* = \emptyset$
10: end if
11: return $[ans, F^*, D^*]$
```

The Algorithm *Is-Feasible* is specified as follows: 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 $\Gamma_i \geq EL[p]$. Let $\Gamma_i(f,d)$ and $l_i(f,d)$ represent the energy level and per-node latency achieved using speed configuration (f,d), respectively. Then, *find_fastest* returns a speed configuration for V_i, $(F^*[i], D^*[i])$ which satisfies:

$$F^*[i] \in \{f_1, f_m\}, D^*[i] \in \{d_1, d_n\} \tag{15}$$

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

$$\forall (f', d') \in \{(f,d) | \Gamma_i(f,d) \geq EL[p])\}, \tag{17}$$
$$l_i(F^*[i], D^*[i]) \leq l_i(f', d')$$

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 $H_i \le \pi, \forall \rho_i$, the constraint (12) is met, hence the answer to problem FS_p is positive (line (6-7)). Otherwise, constraint (12) 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 returns FALSE. 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 $\Gamma_1 \ge \Gamma_2$.*

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, $\forall q \ge p$.

Corollary 1 holds because $EL[p] \ge EL[q]$, 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, $\forall q \ge p$. 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 first half (i.e. $[1, p-1]$) if FS_p has positive answer. Due to the smaller path latency associated with iteration q, $q > p$, any FS_q problem on the second half must also have positive answer, hence it is unnecessary to search that region. Rather, the search on the first half may yield a larger achievable Γ_{min}.
- continue on the second half (i.e. $[p+1, |EL|]$) if FS_p has negative answer. Due to the larger path latency associated with q where $q < p$, any FS_q on the first half will violate the constraint (12), thus having a 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 the paths (line (5)) can be achieved in $O(N)$ time with 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.

B. Speed reduction for nodes on non-critical paths

We note that the function $find\_fastest$ in Algorithm 2 produces node speed assignments that run at the highest possible speed configuration that satisfies $\Gamma_i \ge \Gamma_{min}$. This speed assignment is not always desirable or necessary, and we now present a scheme to reduce speed for the unnecessarily fast nodes, given that Γ_{min} has been maximized by *CHASS*. We first define a *critical path*. Given the speed assignment derived by *CHASS*, a critical path in the tree is any path on which no nodes can further reduce their computation or communication speeds without violating the constraint (12). Note that such critical paths must exist. This is because if the tree does not contain a critical path, then we can further reduce the speed of nodes on any path without violating the constraint (12). This will increase the energy level of these nodes, hence *CHASS* will proceed to a new iteration and find a higher Γ_{min}. This contradicts the optimality of *CHASS* that Γ_{min} is already maximized.

Recall that H_i is the latency for packets from V_i to reach BS, and L_i is the time for V_i to collect packets from all its descendent source nodes. For any node V_i on a critical path, H_i is fixed since we cannot reduce the speed of any node on ρ_i. Since the latency from V_i to BS is fixed at H_i, in order for V_i to collect and forward packets sensed by its descendent nodes to BS within the latency constraint π, any of those packets must reach V_i in no more than time $\pi - H_i$. In other words, the collection latency of V_i, L_i must be smaller than $\pi - H_i$. Therefore $\pi - H_i$ can be interpreted as the latency constraint at V_i, denoted as LC_i.

We refer to the speed reduction scheme as *SpeedReduction*, and give it in Algorithm 3. *SpeedReduction* reduces speeds of nodes based on the speed assignment derived by *CHASS*. Line 1 identifies the path with the largest latency in the tree, denoted as ρ^{max}. Notice that ρ^{max} must be a critical path, and the base station is also on ρ^{max}.

Lines (2-7) iterate over every node V_i on ρ^{max} (from BS down to the leaf node on ρ^{max}) to reduce speeds of their descendent nodes, while ensuring that the resulted collection latency L_i is smaller than the latency constraint at V_i, LC_i. Specifically, in each iteration of the for-loop, procedure *DoSpeedReduction* is called over any immediate children of V_i, except the one resided on ρ^{max}. The procedure *DoSpeedReduction* given in Algorithm 4 executes in recursive fashion. In this way, lines (2-7) will ultimately visit every node in the tree exactly once, in a top-down fashion.

Now we specify Algorithm 4, *DoSpeedReduction*. *DoSpeedReduction* has two inputs, the node V_j whose speed is to be reduced, and the collection latency constraint of its parent, LC_i. *DoSpeedReduction* reduces speed for V_j, while fixing the speeds of V_j's children. That is, $l_j(f,d)$ is to be increased, while L_j remains fixed. Specifically lines (2-3) reduce the speed configuration of any node V_j to a level $(\tilde{f}, \tilde{d})$ that yields the minimum energy consumption $E^c(\tilde{f}, \tilde{d})$ (Eq. 4), among any (f,d)s that give $L_j + l_j(f,d) \le LC_i$. This will yield the maximum energy level increment at these

Algorithm 3 SpeedReduction

```
1: Identify $\rho^{max}$ in the tree.
2: for any node $V_i$ on $\rho^{max}$, from BS down to the leaf node on $\rho^{max}$ do
3:     $LC_i = \pi - H_i$
4:     for any child $V_j$ of $V_i$ do
5:         If $V_j \notin \rho^{max}$ then $DoSpeedReduction(LC_i, V_j)$;
6:     end for
7: end for
```

Algorithm 4 DoSpeedReduction

```
1: Input: $LC_i$, $V_j$
2: $(\tilde{f}, \tilde{d}) = arg\,min\{E^c(f,d) | L_j + l_j(f,d) \leq LC_i\}$
3: Assign $(\tilde{f}, \tilde{d})$ to $V_j$
4: Set $LC_j = LC_i - l_j(\tilde{f}, \tilde{d}) = \pi - H_i - l_j(\tilde{f}, \tilde{d})$
5: for any child $V_k$ of $V_j$ do
6:     $DoSpeedReduction(LC_j,\ V_k)$;
7: end for
```

nodes. The reduction of speed to $(\tilde{f}, \tilde{d})$ will yield a new latency at V_j, $l_j(\tilde{f}, \tilde{d})$.

After the speed reduction at V_j, lines (5-7) continue reducing the speeds of V_j's children V_ks, where the latency constraint LC_j at V_j is set to $\pi - H_i - l_j(\tilde{f}, \tilde{d})$ in line 4. As procedure *DoSpeedReduction* executes recursively until the leaf nodes are reached, it visits every descendent node of V_i, and reduces their speeds.

DoSpeedReduction has the desirable property of reducing the speed of nodes it visits earlier in the procedure. This is desirable because in any tree-structured network high level nodes, nodes which will be visited earlier by the procedure, have more descendant nodes than the low level nodes, and hence are more critical to the system in terms of network connectivity. Therefore, we need to obtain more energy for these nodes in order to prevent them from energy depletion and consequently network partition and service interruption.

Complexity analysis: As mentioned earlier in this subsection, *SpeedReduction* and the recursive procedure *DoSpeedReduction* visit every node in the network exactly once. For each node visited, *DoSpeedReduction* finds the speed configuration $(\tilde{f}, \tilde{d})$ according to line 2 of *DoSpeedReduction* which has time cost of $O(mn)$ given mn speed configurations. Therefore, given N nodes in the network, the total complexity of *SpeedReduction* is $O(mnN)$.

C. Distributed Version

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

The algorithm *DHASS* also proceeds 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 (within the entire tree). The continuous range between the two energy levels is the starting binary search space. It then runs for Y computation rounds, each of which corresponds to one iteration of binary search, and solves one instance of problem FS using the distributed *Is-Feasible*. In each computation round, the midpoint of the search space is used as the 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.

In the initialization round, each node estimates its local Γ^{low}, Γ^{high}, then forwards to its parent as an initialization message. Γ_{low} is calculated using Eq. 4 and 9 while assuming use of the highest speed configuration. Γ_{high} is also calculated using these two equations but assuming the lowest speed configuration. A node receives initialization messages from its children, and compares Γ^{low}, Γ^{high} received to its own values, 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 have received the global Γ^{low} and Γ^{high} values .

The procedure of the distributed *Is-Feasible* is different for non-root and root nodes. Algorithm 5 presents the non-root node case, while Algorithm 6 presents the root node case. Both proceeds iteratively for Y computation rounds. Y is tunable parameter defined by system designers. Each computation round starts by requiring the leaf nodes to send latency messages upwards over the tree. These latency messages will then trigger the computation procedure on the non-leaf nodes.

Lines 3-26 in Algorithm 5 specify the procedure of the distributed *Is-Feasible* in one computation round at the non-root nodes. Line 26 starts a new computation round by directing the execution to line 3. Algorithm 5 has two inputs Γ^{low} and Γ^{high} which have been derived in the initialization round, and outputs the optimal speed configuration and the maximum Γ_{min} found, i.e. MAX_Γ_{min}. It also uses a variable $round$ to keep track of the current computation round. $round$ is initialized to 0.

Algorithm 5 can be explained as follows: Line 3 sets the current computation round. If node V_i is a non-leaf node, then before it starts computation, it must wait to receive latency

Algorithm 5 Procedure of Distributed Is-Feasible at a non-root node V_i

```
1:  Input: $\Gamma^{low}$, $\Gamma^{high}$
    Output: $(F^{opt}[i], D^{opt}[i])$, $MAX\_\Gamma_{min}$
2:  Initialization: $round = 0$
3:  $round = round + 1$
4:  if $V_i$ is a non-leaf node then
5:      Wait to receive $L_j + l_j^{min}$ from every child $V_j$
6:  end if
7:  Compute $L_i = Max.\{L_j + l_j^{min} | V_j \in Children(V_i)\}$
8:  Compute $X = \frac{\Gamma^{low}+\Gamma^{high}}{2}$
9:  $(*, *, l_i^{min}) = find\_fastest(X, V_i)$
10: Send $L_i + l_i^{min}$ to $Parent(V_i)$
11: Wait to receive decision message (containing $ans$) from
    the root
12: if $ans == TRUE$ then
13:     $\Gamma^{low} = X$
14: else
15:     $\Gamma^{high} = X$
16: end if
17: if $round == Y$ then
18:     if $ans == TRUE$ then
19:         $MAX\_\Gamma_{min} = X$
20:     else
21:         $MAX\_\Gamma_{min} = \Gamma^{low}$
22:     end if
23:     $(F^{opt}, D^{opt}, *) = find\_fastest(MAX\_\Gamma_{min}, V_i)$
24:     Return
25: end if
26: Goto line 3
```

Algorithm 6 Procedure of Distributed Is-Feasible at the root

```
1: Wait until $L_j + l_j^{min}$ has been received from all children
2: $L_{tot} = Max.\{L_j + l_j^{min} | V_j \in Children(Root)\}$
3: if $L^{tot} \leq \pi$ then
4:     $ans = TRUE$
5: else
6:     $ans = FALSE$
7: end if
8: Disseminate $ans$ across the tree
```

messages from each of its children V_j (line 4-6). A latency message contains $L_j + l_j^{min}$ computed by its child V_j. Using the received $L_j + l_j^{min}$, V_i computes its collection latency L_i using Eq. 7 in line 7. If V_i is a leaf node which has no child, then it has $L_i = 0$ and does not wait for latency messages. Next in line 9, V_i derives the smallest per-node latency l_i^{min} while satisfying $\Gamma_i \geq X$, by calling function $find\_fastest$ specified in 5-A. $X = \frac{\Gamma^{low}+\Gamma^{high}}{2}$ is the input energy level in the current computation round (line 8). Then V_i adds up l_i^{min} to L_i, sends to its parent (line 10), and waits to receive decision message to be disseminated from the root (line 11).

In Algorithm 6, the specification of the root-side procedure of distributed *Is-Feasible*. As each node V_i receives $L_j + l_j^{min}$ from its children, it computes and sends its own $L_i + l_i^{min}$, the root will compute the total collection latency L_{tot} after receiving $L_j + l_j^{min}$ from all its children (line 1-2). In line 3-7, the root compares L_{tot} to the latency constraint π. If $L_{tot} \leq \pi$, the root sets the answer to the associated problem FS in the current round, i.e. ans as TRUE; otherwise it sets ans to FALSE. Finally, the root disseminates ans across the tree (line 8).

Returning to Algorithm 5, when V_i receives the decision message from the root (line 11), it updates the binary search range (represented by Γ^{low} and Γ^{high}) for the next computation round. This is based on the answer ans enclosed in the decision message. If $ans == TRUE$, V_i sets Γ^{low} to X which directs the binary search to the higher half of the current search range, otherwise it sets $\Gamma^{high} = X$ which directs the search to the lower half (line 12-16). Then V_i starts a new computation round by directing the execution to line 3. Finally, if the current computation round is the Y^{th} round (line 17), then V_i determines MAX_Γ_{min} as follow:

- If $ans == TRUE$, MAX_Γ_{min} is set to be the X in the Y^{th} round (line (18-19)).
- Otherwise, MAX_Γ_{min} is set to be the Γ^{low} in the Y^{th} round (line (20-22)).

Using the derived MAX_Γ_{min}, V_i calls function *find_fastest* to determine the optimal speed configuration $(F^{opt}[i], D^{opt}[i])$ (line 23), and terminates (line 24).

We can now present Theorem 1, showing that 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. As seen from Theorem 1, using a larger Y for *DHASS* achieves closer performance to the optimal, however this comes at cost of higher complexity shown as follow.

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 initialization 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

A. 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. Water meters are increasingly used to provide customers and companies instantaneous pricing information, measurements of water quality, detecting the presence of containments. Further, these meters can be equipped with actuators to deal with emergency situations such as contamination and breakage. 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.

We have developed simulation software upon TOSSIM: the standard high-fidelity WSN simulator, combined with EPANET [26], a public domain, water distribution system modeling program. 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 36, 64 and 100 node residential water distribution topology. The topology is derived from an existing suburban area.

Due to a typical consumer repetitive water usage pattern with a cycle of 6 hours, we fixed the harvesting horizon at H=6 hours. The 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 harvesting profile based on the observed water usage at the customers. The frame period is set to π=$240ms$. Both *CHASS* and *DHASS* were implemented in our simulation along with the speed reduction schemes. As noted in Section 2, due to architectural and performance objective issues, there are no algorithms that are comparable to ours. Therefore, we compare HASS algorithm to an energy harvesting unaware scheme, NPM (No Power Management) scheme which simply maximizes performance by using the highest frequency and modulation levels on all nodes, in order to meet, whenever possible, the data collection timing constraints. Our experiments considered two basic application types: applications that support *complete-aggregation* and applications those 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 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=$[0, 3000000]$ cycles. The 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 [8]. The iMote-2 platform has a Intel Xscale PXA27x CPU [16] and a ChipCon CC2420 radio [23]. The frequency and power specification of PXA27x processor is given in Table (II). Although CC2420 does not support DMS, we model a DMS-capable radio by assuming four modulation levels: $b = \{2, 4, 6, 8\}$ [28] which gives four communication speeds, $d = \{125, 250, 375, 500\}$ kbps (symbol rate $R = 62.5k$ [23]). The radio power levels are calculated using the CC2420 specification [23] and the radio energy model used in [28]. 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.

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

TABLE II: Specification of Intel Xscale Pxa27x

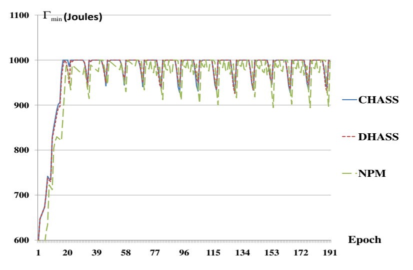
Nodes operate in either *normal* or *emergency* mode. We represent the emergency mode by scaling up the frame-based workload by w times upon the normal mode, where w is a tunable parameter. This reflects the fact that nodes will need to perform additional duties during those times. By using EPANET functionality we simulate emergency scenarios by introducing contaminant into the system at random times. This can, for instance, represent a terrorist attack on the water supply. As contaminant spreads, the water quality in the residences decreases and is finally detected by sensor nodes. A sensor node then switches to emergency mode and executes additional workload over a series of epochs, until the water quality returns to normal. We consider three different types of emergency scenarios. The first type is a *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. The third mode is *area instant* (INST) attack. Under this scenario a large contiguous area of the network is affected.

B. Results

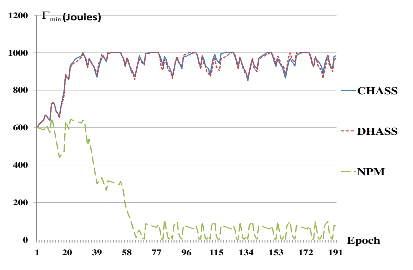
In *CHASS* scheme, the set EL contains 2400 elements, given 6 CPU frequencies, 4 modulation levels, and 100 nodes. In *DHASS* scheme, we set the number of search iterations to be $\log(|EL|) \approx 11$. We evaluated the performance of our algorithms under normal and all three emergency modes. Each scenario was tested using complete aggregation and non-aggregation. We also varied emergency workload levels.

1) Non-aggregating applications: In Fig. 2(a-c), we compare different schemes in terms of the achieved Γ_{min}, while assuming non-aggregating applications. We fix the emergency level w at 3.0, meaning that the emergency workload is three times the normal workload. In normal mode (Fig. 2(a)), the Γ_{min} value can be seen to vary semi-repetitively. 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. 2(b,c), the significantly increased workload demand (in RAND and SPRD emergency modes) 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. Due to space limitations, we have omitted the Γ_{min} plot under INST mode.

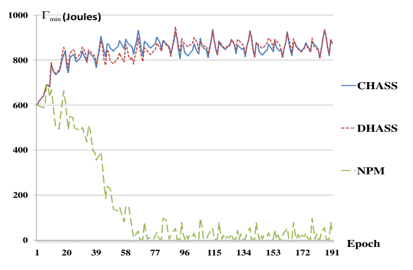
As seen from all above figures, in the normal and emergency modes, the *CHASS* scheme achieves the highest Γ_{min}, followed by *DHASS* with slightly lower Γ_{min}, and then *NPM*. In normal mode (Fig. 2(a)), *NPM* scheme also achieves a high Γ_{min}. This is because the harvested energy is much larger than energy demand in normal mode. As workload demand increases in emergency modes (Fig. 2(b,c)), the performance of *NPM* drops drastically; its achieved Γ_{min} drops to zero around the 58^{th} epoch for all three emergency modes. In comparison both *HASS* approaches achieve much higher Γ_{min} than *NPM* (Fig. 2(b,c)). This is because the *HASS* method



a. Normal



b. RAND mode



c. SPRD mode

Fig. 2: Min. energy level, non-aggregation

allows energy-rich nodes run at faster speeds and therefore permits to allow the harvesting-poor to slow down.

Notice that *DHASS* achieves a Γ_{min} level very close to that of *CHASS*. In normal mode (Fig. 2(a)), the achieved Γ_{min} of *CHASS* and *DHASS* almost overlap. In emergency modes, the performance difference between them increases 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.

w	1	1.5	2.0	2.5	3.0
$RAND$	0%	0%	27.2%	31.6%	33.4%
$SPRD$	0%	0%	38.7%	41.6%	43.2%
$INST$	0%	0%	32.8%	53.9%	55.2%

TABLE III: Percentage of interrupted nodes: NPM, Non-aggregation

We then conducted a *stress test* over the system while using different schemes. 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 terms of the percentage of nodes that are interrupted by energy depletion.

Table III gives the percentage of interrupted nodes in all three emergency modes under various emergency intensities, using *NPM* scheme. As seen from Table III, as emergency intensity increases, the percentage of interrupted nodes increases noticeably in all modes when using *NPM*, implying the low resilience of the harvesting-unaware NPM scheme to emergency situations. We then found that though there are very limited number of nodes who ran out of energy in the network, large amount of nodes are also interrupted. This is because the depleted nodes are mostly close to BS which have large group of descendent nodes and thereby high workload demands. The depletion of these nodes interrupts all their descendent nodes. While using *CHASS* and *DHASS*, the same increase in emergency intensity interrupts no node at all in the network. The results of the stress test demonstrates the benefit of our harvesting-aware approaches in mitigating the impact of emergencies over the system.

2) Aggregating applications: For aggregating applications, we repeat the same set of experiments conducted for non-aggregating applications. In Fig. 3(a-c), we also fix the emergency intensity at $w = 3.0$ and plotted the Γ_{min} achieved by

w	1	1.5	2.0	2.5	3.0
$RAND$	0%	0%	0%	0%	26.9%
$SPRD$	0%	0%	0%	0%	49.1%
$INST$	0%	0%	0%	0%	47.5%

TABLE IV: Percentage of interrupted nodes using NPM - aggregation

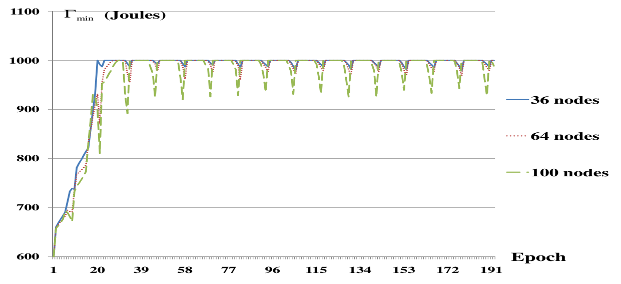
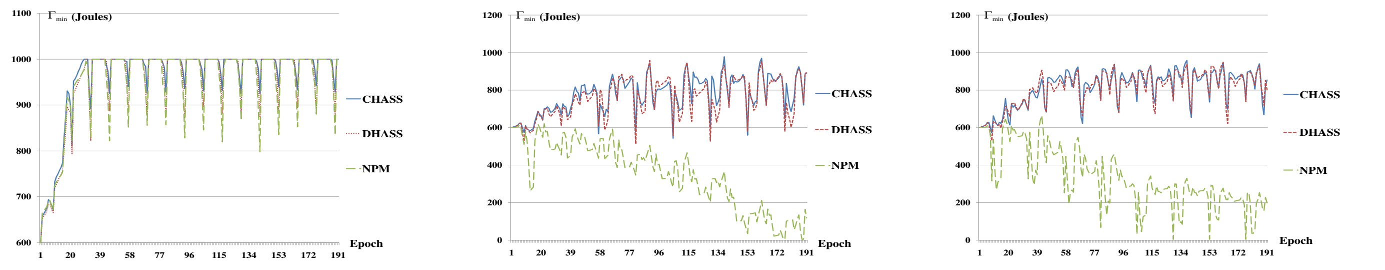


Fig. 4: Min. energy level as a function of different network sizes - Normal, complete aggregation

using different schemes. In all the modes, *CHASS* and *DHASS* schemes again achieve much higher Γ_{min} than *NPM*. *DHASS* performance is very close to the *CHASS* scheme. As seen from all figures, the Γ_{min} achieved by our *HASS* approaches never drops to zero, while the achieved Γ_{min} by *NPM* schemes drops to zero many times. We then repeat the stress test. No nodes are interrupted when *HASS* scheme is used, while large percentage of nodes are interrupted when emergency intensity increase to $w = 3.0$ for the INST and SPRD modes.

We next present in Fig. 4 the achieved Γ_{min} using *CHASS* scheme for networks of different sizes (with 36, 64 and 100 nodes). As seen from the figure, the 36-nodes network has the highest Γ_{min}, followed by 64-nodes and 100-nodes networks. Although the difference of Γ_{min} among different networks is small, we observe that the smaller the network, the higher the value of Γ_{min}. This is because given the fixed end to end latency constraint (240 ms), nodes in large networks have more hops in a path to BS, thus the intermediate nodes have less time slack for processing and communication, thereby must use high computation and communication speeds.

In Fig. 5, we assume inaccurate prediction of harvested energy and workloads, and plot Γ_{min} values for SPRD attack, at emergency level $w = 3.0$ for aggregating applications. That is, the actual amount of energy harvested is up to 25% higher or lower than the harvesting prediction. Also, the actual



a. Normal b. SPRD mode c. INST mode

Fig. 3: Min. energy level, complete aggregation

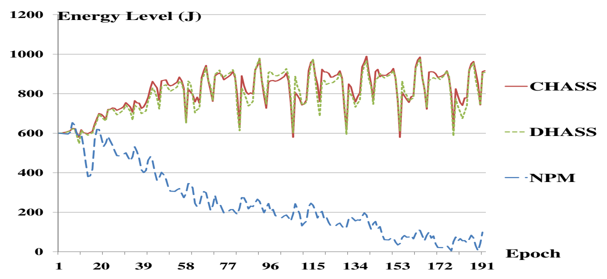


Fig. 5: Min. energy level with probabilistic harvesting profile and workloads - SPRD, complete aggregation

computation and communication workloads, C and M are 25% lower than the workload prediction, as both C and M are the worst-case workloads and tasks may complete earlier. As seen from the figures, our HASS approach still outperforms NPM under assumption of probabilistic energy harvesting and workloads, however the performance difference decreases as compared to Fig. 3(b) which share the same experiment setting but with accurate harvesting and workload predictions. This is because, due to prediction inaccuracy, our HASS schemes outputs sub-optimal speed assignment, while NPM scheme is not affected by prediction inaccuracy as it always uses the maximum speed configuration.

Finally, we calculated the communication overhead of *DHASS*. In our experiments, *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 $12/(3750 + 12) \approx 0.32\%$. This implies that our *HASS* approaches have also the benefit of high efficiency.

7. Conclusions

This paper presented an epoch-based approach for energy management in performance-constrained WSNs that utilize energy harvesting. 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 formulated our problem as an optimization problem, and solved it with centralized and distributed algorithms. Through simulation we show our algorithms achieve significantly higher performance than a baseline approach under both normal and emergency situations.

Appendix

Proof of Lemma 1:

Proof: 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, we must have $S^* = S^N$.
- In case that S^N leads to $\Gamma_i > \Gamma^{max}$, $\exists V_i$, we claim that the computation and communication 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:

Proof: 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 $l^{min}_{i,1}$ and $l^{min}_{i,2}$ 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 $l^{min}_{i,1} \geq l^{min}_{i,2}$.

When the FS problem is solved with Γ_1 as input, (f_1, d_1) yields $l^{min}_{i,1}$, 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 $l^{min}_{i,2} = l^{min}_{i,1}$, while satisfying $\Gamma_i(f_2, d_2) \geq \Gamma_2$ (because $\Gamma_i(f_2, d_2) = \Gamma_i(f_1, d_1) \geq \Gamma_1 \geq \Gamma_2$). Notice that in many cases, (f_2, d_2) will yield an even smaller $l^{min}_{i,2}$. ■

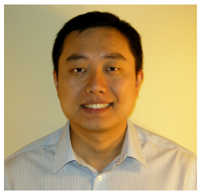
Proof of Theorem 1:

Proof: 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] Hanzalek, Z., Jurcik, P., "Energy Efficient Scheduling for Cluster-Tree Wireless Sensor Networks With Time-Bounded Data Flows: Application to IEEE 802.15.4/ZigBee", IEEE Trans. on Industrial Informatics, vol.6, no.3, pp.438-450, Aug. 2010

[2] G. Anastasi, M. Conti, M. Di Francesco, "Extending the Lifetime of Wireless Sensor Networks Through Adaptive Sleep ," IEEE Trans. on Industrial Informatics, vol. 56, no. 3, pp. 351 - 365, July 2009.
[3] Y. K. Tan, S. K. Panda, "Energy Harvesting From Hybrid Indoor Ambient Light and Thermal Energy Sources for Enhanced Performance of Wireless Sensor Nodes," IEEE Trans. on Industrial Electronics, vol. 58, no. 9, 2011
[4] S. E. Yoo, P. K. Chong, D. Kim, Y. Doh, M. L. Pham, E. Choi, J. Huh, "Guaranteeing Real-Time Services for Industrial Wireless Sensor Networks With IEEE 802.15.4 ," IEEE Trans. on Industrial Electronics, vol. 57, no. 11, pp., Nov. 2010.
[5] Allen, G. W., Haggerty, S. D. and Welsh, M., Lance: Optimizing high-resolution signal collection in wireless sensor networks, In the 6th ACM conference on Embedded network sensor systems, Raleigh, NC, 2008
[6] Aydin, H. et al., Power-aware scheduling for periodic real-time tasks, In IEEE Trans. on Computers, vol 53, page 584-600, 2004
[7] Challen, G. W., Waterman, J. and Welsh, M., Integrated distributed energy awareness for wireless sensor networks, In the 7th ACM Conference on Embedded Networked Sensor Systems (sensys'09), Berkeley, CA, 2009
[8] Crossbow Technology, iMote2 datasheet, www.xbow.com
[9] R. S. Liu, K. W. Fan, Z. Z. Zheng, Sinha, P., "Perpetual and Fair Data Collection for Environmental Energy Harvesting Sensor Networks", IEEE/ACM Trans. on Networking, vol.19, no.4, pp.947-960, Aug. 2011
[10] Mainland, G., Parkes, D. C. and Welsh, M., Decentralized, adaptive resource allocation for sensor networks, In the 2nd conference on Symposium on Networked Systems Design and Implementation (NSDI'05), Boston, MA, 2005
[11] Heo, J., Hong, J. and Cho, Y., EARQ: Energy aware routing for real-time and reliable communication in wireless industrial sensor networks, In IEEE Trans. on Industrial Informatics, vol 5, no 1, pp. 3-11. Feb. 2009
[12] Hwang, W. L., Fei, S. and Chakrabarty, K., Automated design of pin-constrained digital microfluidic arrays for lab-on-a-chip applications, In ACM Design Automation Conference, San Francisco, CA, 2006
[13] Kansal, A. et al., Power management in energy harvesting sensor networks, In ACM Trans. on Embedded Computing Systems, vol 6 no 4, Sept. 2007
[14] Liu, S., Qiu, Q. and Wu, Q., Energy aware dynamic voltage and frequency selection for real-time systems with energy harvesting, In Design Automation and Test in Europe, Munich, Germany, 2008
[15] Madden, S. et al., TAG: a Tiny AGgregation service for ad-hoc sensor networks, In the USENIX Symposium on Operating Systems Design and Implementation (OSDI'02), Boston, MA, 2002
[16] Marvell Technology Group Ltd., Xscale pxa27x data sheet, www.intel.com/design/intelxscale
[17] Moser, C., Chen, J. and Thiele, L., An energy management framework for energy harvesting embedded systems, ACM Journal on Emerging Technologies in Computing Systems, vol. 6 no. 2, June 2010
[18] Moser, C., Thiele, L., Brunelli, D., Benini, L., "Adaptive Power Management for Environmentally Powered Systems," IEEE Trans. on Computers, vol.59, no.4, pp. 478-491, April 2010
[19] Moser, C. et al., Real-time scheduling with regenerative energy, In Euromicro Conference on Real-Time Systems (ECRTS'06), Dresden, Germany, 2006
[20] Paek, J. et al., The TENET architecture for tiered sensor networks, In ACM Trans. on Sensor Networks, vol. 6, no. 4, 2010.
[21] Shah, P. et al., Power management using zigBee wireless sensor network, In International Conference on Emerging Trends in Engineering and Technology (ICETET'08), Nagpur, India, 2008
[22] Sudha Anil Kumar, G., Manimaran, G. and Wang Z., "End-to-end energy management in networked real-time embedded systems," IEEE Trans. on Parallel and Distributed Systems, pp. 1498-1510, Nov., 2008
[23] Texas Instrument, CC2420 data sheet, docs.tinyos.net/index.php/CC2420
[24] Tidwell, T. et al., Optimal time utility based scheduling policy design for cyber-physical systems, Department of Computer Science, Washington University, TR 2010-27, May 2010
[25] US Department of Energy, National SCADA test bed study of security attributes of smart grid systems current cyber security issues, April 2009
[26] US Environmental Protection Agency, EPANET 2.0, Water supply and water resources, www.epa.gov/nrmrl/wswrd/dw/epanet.html, 2010
[27] Verissimo, P. et al., Designing modular and redundant cyber architectures for process control: lessons learned, In the 42nd Hawaii International Conference on System Sciences (HICSS'09), 2009
[28] Y. Yu, Prassana, V.K., Krishnamachari, B., "Energy Minimization for Real-Time Data Gathering in Wireless Sensor Networks", IEEE Trans. on Wireless Communications, vol.5, no.11, pp.3087-3096, Nov. 2006
[29] Zamora, N. H., Kao, J. C. and Marculescu, R., Distributed power management techniques for wireless network video systems, In Design Automation and Test in Europe, Nice Acropolis, France, 2007
[30] Yanjun Li, Chung Shue Chen, Ye-Qiong Song, Zhi Wang, Youxian Sun, "Enhancing Real-Time Delivery in Wireless Sensor Networks With Two-Hop Information ," IEEE Trans. on Industrial Informatics, vol. 56, no. 2, pp. 113 - 122, April 2009.
[31] L. LoBello, E. Toscano, "An Adaptive Approach to Topology Management in Large and Dense Real-Time Wireless Sensor Networks ," IEEE Trans. on Industrial Informatics, vol. 56, no. 3, pp. 314 - 324, July 2009.
[32] Bernat, G.; Colin, A.; Petters, S.M.; , "WCET analysis of probabilistic hard real-time systems," IEEE Real-Time Systems Symposium (RTSS'02), Austin, TX, 2002
[33] A. Ravinagarajan, D. Dondi, and T Simunic Rosing. DVFS based task scheduling in a harvesting WSN for structural health monitoring. In the Conference on Design, Automation and Test in Europe (DATE'10). Leuven, Belgium
[34] Shaobo Liu, Qing Wu, and Qinru Qiu. An adaptive scheduling and voltage/frequency selection algorithm for real-time energy harvesting systems. In Proceedings of the 46th Annual Design Automation Conference (DAC'09), New York, NY. 2009
[35] Clemens Moser, Jian-Jia Chen, and Lothar Thiele, Power management in energy harvesting embedded systems with discrete service levels. In Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design (ISLPED'09), New York, NY. 2009



Bo Zhang Bo Zhang receives his BS. degree from Huazhong University of Science and Technology, Wuhan, China; and MS. degree from University of Cincinnati, Cincinnati, OH, both in computer science. He is currently a PhD student in computer science department of George Mason University, Fairfax, VA. His research interests include wireless sensor networks, low-power embedded systems.

Robert Simon Dr. Robert Simon is an Associate Professor of Computer Science at George Mason University in Fairfax, VA. He received a B.S. in History and Political Science from the University of Rochester, and a Ph.D. in Computer Science from the University of Pittsburgh. His research interests include embedded systems, wireless and mobile computing, distributed systems and performance modeling and analysis and distributed computing. He has published over 80 peer-reviewed journal and conference papers on these topics. His research has been supported by a number of agencies, including NSF, DARPA, the US Department of Defense and private industry.

Hakan Aydin Hakan Aydin received the BS and MS degrees in control and computer engineering from Istanbul Technical University in 1991 and 1994, respectively, and the PhD degree in computer science from the University of Pittsburgh in 2001. He is currently an associate professor in the Computer Science Department at George Mason University, Fairfax, Virginia. He has served on the program committees of several real-time and embedded systems related conferences and workshops. He was the Technical Program Committee Chair of 2011 IEEE Real-time Technology and Applications Symposium (RTAS'11). He received the US National Science Foundation (NSF) Faculty Early Career Development (CAREER) Award in 2006. His research interests include real-time embedded systems, low-power computing, and fault tolerance. He is a member of the IEEE.