Node-level Energy Management for Sensor Networks in the Presence of Multiple Applications

Athanassios Boulis and Mani B. Srivastava

Networked and Embedded System Laborarory (NESL), EE Department, University of California at Los Angeles, email: { boulis, mbs }@ee.ucla.edu

Abstract

Energy related research in wireless ad hoc sensor networks (WASNs) is focusing on energy saving techniques in the application-, protocol-, service-, or hardware-level. Little has been done to manage the finite amount of energy for a given (possibly optimally-designed) set of applications, protocols and hardware. Given multiple candidate applications (i.e., distributed algorithms in a WASN) of different energy costs and different user rewards, how does one manage a finite energy amount? Where does one provide energy, so as to maximize the useful work done (i.e. maximize user rewards)? We formulate the problem at the node-level, by having system-level "hints" from the applications. In order to tackle the central problem we first identify the energy consumption patterns of applications in WASNs, we propose ways for real-time measurements of the energy consumption by individual applications, and we solve the problem of estimating the extra energy consumption that a new application brings to a set of executing applications. Having these tools at our disposal, and by properly abstracting the problem we present an optimal admission control policy and a post-admission policing mechanism at the node-level. The admission policy can achieve up to 48% increase in user rewards compared to the absence of energy management, for a variety of application mixes.[*]

1. Introduction

Wireless Ad-hoc Sensor Networks (WASNs) are the main representative of pervasive computing in large-scale physical environments. Networks of a large number of cheap, small-form, wireless devices, embedded in the physical world, may be used for applications such as premise security and surveillance, environmental habitat monitoring, condition-based maintenance, battlefields etc. Most of the research work in WASNs revolves around energy, focusing predominately in energy saving problems. The energy source in each sensor node is limited to the initial battery charge. Replenishing the battery charge is infeasible or so costly that overcomes any benefits drawn from the WASN. Sustainable energy sources, such as solar power, ambient vibrations, acoustic signals, have yet to be proven realizable and efficient in today's WASNs [1][5][8]. Consequently, most research efforts use energy consumption as one of their efficiency metrics. Applications, protocols, services, and hardware are designed to reduce the energy consumption while maintaining their functionality. While these efforts are absolutely necessary for the evolution of WASNs into something more than academic research, they are not the only viewpoint of energy related issues in WASNs.

WASNs are currently envisioned to have long life spans, servicing many transient users with different needs. This vision is currently supported by a series of frameworks that try to make WASNs dynamically programmable and generally open to transient users [4][6][9]. "Multiple different requests for physical information, arriving at different times", translates into "multiple different applications running concurrently in the network while multiple requests for execution of new applications are constantly received". This setting poses the question: *Given a finite energy amount and an unknown sequence of application requests (chosen from a set of candidate applications with known occurrence probabilities, energy costs and user rewards/penalties), how does one accept/reject applications into the network, in order to maximize overall user rewards?* The terms: "requests for information", "application requests", "applications", and "distributed algorithms" are used interchangeably in the text. These are the items handled (i.e., undergo admission control and policing) in order to maximize rewards for all users.

This is an operations research problem in its core (as so many other problems in engineering). The difficulty lies mainly on the formulation of the problem. Do we consider the finite energy amount at the node or the system/network level? What is an application's energy cost and how is it measured? How are user rewards defined? Section 2 argues that a pure system-level approach, although yielding optimal results, is unrealistic, as it requires from each application to have full knowledge of every other application in the WASN (which is contradictory to the notion of transient WASN users), or pay huge traffic

[*] This work was supported in part by the Office of Naval Research under the AINS research program.

overheads. We delve into this issue on section 2 and formulate the problem at the node level without requiring by the applications to explicitly manage the system's energy. Applications just provide our energy management mechanism with attributes about their energy consumption and rewards. Admission control and policing is carried out automatically by our system maximizing user rewards in the average sense at the node level. Applications still change their behavior at the system-level, based on acceptance/rejection replies received by nodes, not based on arbitrary negotiations with other applications.

After the formulation of the problem in section 2, we present solutions for two sub-problems of our energy managing mechanism in section 3. More specifically, the first sub-problem is to estimate the energy cost of an application, based on the provided attributes and behavior of the applications already running in the node. The second sub-problem is to actually measure the energy cost of an admitted application, based on its behavior and behaviors of other applications currently executing in the same node.

Finally in section 4, after properly abstracting the problem in the previous sections we present an optimal admission control policy and a post-admission policing mechanism. The solution is based on the solution of the Dynamic Stochastic Knapsack Problem (DSKP) [7][10]. Particular consideration was given to the computational complexity of the policy, since sensor nodes are computationally restricted. The admission and eviction decisions take minimal time as all computationally intensive quantities of the DSPK problem can be computed off-line in our case. Our mechanism achieves up to 48% increase in user rewards compared to the absence of energy management, for a variety of application mixes. Section 5 concludes the paper.

2. Problem Formulation

The problem would be optimally considered at the system level. After all, a WASN is a distributed hardware platform with an arbitrary set of *distributed* applications executing on it. Therefore, only a viewpoint that considered the system as a whole would provide the optimal answer. We wish to show though that a purely system-level approach is unrealistic. Consider the following properties of WASN applications: 1) An application can be distributed in many different sets of nodes and still work. Of course, with a varying distribution set, the quality of the returned information (reward) and the energy spent vary. 2) When parts of two or more applications are executing in one node, the total energy consumption is not the sum of the individual energy consumptions due to sharable modules and services. Imagine now two applications already running in the WASN with 5 different distribution choices each. Imagine also a new application with only 3 different distribution possibilities. Even with these moderate numbers we have $5\cdot5\cdot3=75$ different configurations to check in order to find the globally optimum solution. Even if a protocol is created to handle the negotiations between the applications in a standardized way, the overhead traffic to find the solution would greatly surpass any benefits drawn from the energy management algorithm. For every configuration, every involved node would have to send reward and energy information to a central place.

We need a more-modular/less-holistic solution that does not require from applications to explicitly negotiate *in a huge solution space* in order to manage the available energy. We advocate a node-level solution with system level "hints" from the applications. More specifically, when some portion of a distributed application (i.e. a piece of code that executes in one node, henceforth named ***task***) wishes to be transferred and executed in a specific node, it needs to carry some attributes with it. The attributes along with the current node state $\mathcal{N}$ (e.g., remaining energy, parameters of tasks already running in the node) will determine the task's admission (or not) based on some policy $\mathcal{P}$. The energy management system proposed here, fits in a greater framework for programmable WASNs described in [4]. The interested reader can find information on distributed application creation and deployment in [4]. In our energy management system there are three types of attributes: Energy attributes $\{e_{attr}\}$, reward attributes $\{r_{attr}\}$, and policing attributes $\{p_{attr}\}$. These are used in order to derive the following quantities: *energy_cost* $= f_1(\{e_{attr}\}, \mathcal{N})$, *reward* $= f_2(\{r_{attr}\})$, *admission_decision* $= \mathcal{P}($*energy_cost, reward, remaining_energy*$)$. $\{p_{attr}\}$ are used for policing after a task has been admitted. In the next three subsections we will define these attributes.

2.1. Energy attributes

The first step in defining the energy attributes is to identify the energy consumption patterns of applications in WASNs. How is energy spent by applications? At the lower level, energy is spent on hardware operations (e.g., instructions executed in the CPU, bits transmitted/received by the radio, samples acquired by the sensing device). However, one can view energy consumption at higher-level operations. For instance, there is an average energy consumption associated with the transmission of a packet of size L, using a particular protocol. To calculate this energy one needs to know the protocol, (i.e., header overhead, retransmission policies), the radio, and the traffic characteristics at the time of transmission. Since the last term is dynamic we can calculate the energy spent in the average sense. To move even higher in the levels that one can view energy consumption, consider the following example. Services common to sensor nodes, like neighborhood discovery and localization, have an average energy consumption. According to the network topology and other service parameters (e.g. accuracy in localization, n in "discover n-hop neighbors"), a number of messages is exchanged. The communication and processing of the messages consume the energy.

Viewing the energy consumption at higher-levels is beneficial for our purposes. Usually, applications built for WASNs use services and protocols already provided by sensor nodes [4][6][9]. Describing the energy consumption in low level attributes, such as transmitted bits, reception time, and computation time, would be impossible since the application does not have any knowledge of sensor node resident services and protocols. On the other hand, it is convenient for a task to describe the usage of the node's modules and services, and let the individual modules/services derive the energy consumption. For example, a task may specify that it needs to know the node's location with 1cm accuracy. If the localization service already has this information then no energy will be spent. The local service can also know that if no information is available x Joules of energy will be spent, while if the location is already known with a 2cm accuracy y Joules will be spent (x>y). So, calculating the energy cost of a task cannot rely only upon the task, but it should be also affected by the node's state, network conditions, and other tasks running on the node.

Following the above rationale, the energy attributes in our system are a list of module/service names, each name followed by some parameters that declare the module/service usage. The total task time is also given. For instance $\{e_{attr}\}$ could be: {radio_receive 3sec, radio transmit broadcast 1000bytes routing 200bytes, sensor sample_rate 1Hz size_sample 2bytes, CPU 10000 instructions, localization service accuracy 5cm, total task time 60sec}. The values of the energy attributes are set by the programmer during the creation of the application. The programmer has full knowledge of the tasks created by the application, their average behavior, as well as all the modules and services that a task can use.

It is important to note here that some modules/services (henceforth named *devices*) are ***sharable***. This means that the devices can be used concurrently by multiple tasks while consuming the same energy as being used by one task. We already saw an example of a sharable device in the localization service. The location needs to be discovered with the highest required accuracy to satisfy all tasks. Other examples of sharable devices are: the radio in receive-mode, the sensing device, the real-timer service, and the neighborhood discovery service. Examples of non-sharable devices are: the radio in transmit-mode, and the CPU. Having sharable devices creates the problem of determining the energy cost of a task given a set of other tasks already executing in the node. For each sharable device i, we have to calculate the Added Energy Load (AEL_i) that this task is bringing to the tasks already running. The sum of all AEL_i will give us the total AEL, i.e., the energy cost of the task. AEL_i is a device specific function. Some devices (e.g., localization service) may have a pre-computed reference table to determine the AEL based on the usage parameters provided by the new under-admission task and current usage by other tasks. Many devices though, can compute AEL on-the-fly. Examples of such devices are: the radio in receive-mode and the sensing device. We present a way to calculate AEL (or more accurately E[AEL]) for such sharable devices in section 3.

In conclusion, $\{e_{attr}\}$ is device dependent and can be written as a set of sets of parameters (one set for each device): $\{\{e_{attr\_dev1}\} \dots \{e_{attr\_devn}\}\}$. Given these attributes the energy cost of a task is $energy\_cost = f_1(\{e_{attr}\}, \mathscr{N}) = \sum_{i:sharable\ device} AEL_i(\{e_{attr\_devi}\}, N) + \sum_{i:non\ sharable\ device} AEL_i(\{e_{attr\_devi}\})$

The AEL_i functions for non-sharable devices are simply calculating the energy consumption of the device due to a specific task. Since the devices are not sharable, the state of the node $\mathscr{N}$ is not included in the AEL_i arguments. As stated, in section 3 we will define AEL for a simple class of sharable devices.

2.2. Reward attributes

Defining reward attributes is a particularly difficult task in our scheme. This is due to the fact that even though reward is a system-level quantity (i.e., the user receives the needed information [reward] because of the collective behavior of the involved nodes), we are called to derive user rewards at the node level. Generally speaking, there should be a reward associated with the admission and completion of an application. At the same time, if a user request is denied service, there is a penalty associated with such action. We define the penalty for denying a request/application to be executed to be equal to its reward.

We begin by defining reward in the system-level. By default, each user request has a reward equal to 1 if carried out as specified. This assumes that all users are equal, thus all data returned due to individual requests have the same reward. Given an external user differentiation policy, we can accordingly differentiate rewards for different users. If the application cannot be admitted as is, the application might provide an alternate distribution and/or an alternate algorithm that achieves less accurate information. The reward of the application must be reduced proportionally to the achieved accuracy. Note that such a procedure (of reward reduction) is initiated by the application at the event of non-admission (i.e., at least one of the application's task was not spawned in the targeted node). The reward reduction procedure essentially reinitiates another application, which may range from a simple redistribution of the existing tasks to the execution of a new algorithm. The initial application's tasks that were previously admitted in their targeted nodes are either canceled (if they are no longer needed by the reduced-reward application), or their reward is reduced (in order to correspond to the reward of the reduced-reward application).

Moving to the node level we have to answer the question: How does the system-level reward influence the reward at the node level? One might suggest the system level reward to be separated among the different tasks (i.e., the parts of the application running at different nodes). This approach is wrong though. Consider the following scenario: We have two applications A and B of equal importance (reward =1). Application A is distributed in 2 nodes and application B in 100 nodes. If we were to separate the unit reward equally, the tasks of application A would get 0.5 each, while the tasks of application B would get only 0.01 each. Consequently, the tasks of application B would have a much smaller probability to be executed (because they seem to offer so little) while at the system level the two applications are equivalent. Therefore, to avoid such problems *we define the node level reward to be the system level reward*.

From the above discussion we conclude that there are two reward attributes $\{r_{attr1}, r_{attr2}\}$ and they are defined as: $\{r_{attr1}\equiv$user priority$\in(0,\infty)$ (=1 by default for all users), $r_{attr2}\equiv$application reward$\in(0,1]$ (=1 for initial request [full accuracy information])$\}$; r_{attr1} shows the importance of the user; r_{attr2} shows how far away is this application from its initial desired information (due to rejections of higher accuracy requests). These two attributes are "hinting" about the system level behavior of the application. The reward of a task is $reward = f_2(\{r_{attr}\}) = r_{attr1} \cdot r_{attr2}$.

2.3. Policing attributes

Finally, we need some attributes to specify some quantities for policing issues. Once a task has been admitted we need to make sure that its execution is beneficial to the user. After all, the admission was based on estimated values. In short we need to measure its true energy costs and make sure that together with its rewards, the task is beneficial to the user. To perform policing we need to know what is the proper time interval to measure energy costs (i.e., we want to take an interval large enough to avoid transient effects), as well as the granularity of reward return (i.e., is the reward returned as a whole at the end of the task execution, or is it gradually given as the task runs). Both these quantities are application specific, so we require the task to provide them in the form of policing attributes: $p_{attr1}\equiv$energy-measurement time interval, $p_{attr2}\equiv$minimum return-reward time interval.

3. Calculating the AEL and Measuring Energy Costs During Execution

In this section we will examine two sub-problems of the general energy management mechanism. We referred to the first problem in section 2.1; it is the AEL calculation problem. The second problem is the real-condition measurement of energy costs of the admitted tasks.

3.1. Calculation of the AEL

Consider a class of sharable devices with the following simple sharing rule: Devices in this class are used for amounts of time; if two or more tasks use the same device concurrently the energy consumption is the same as only one task uses the device. Examples of such devices are: the radio in receive-mode and the sensing device. Given that a task specifies the time that is using such a device as well as the total task time in $\{e_{attr}\}$, we can know the fraction of total time the device is used by the task. Having this fraction and the usage fraction by all the currently executing tasks, how do we calculate E[AEL] (i.e., the mean added energy load) that the new task is bringing to the set of old tasks?

The core of the problem can be identified as following. Given that all the existing tasks in a node are using a device a fraction of the unit time p, and the new under-admission task is using it a fraction of the unit time q, how much more (as a fraction of the unit time) will the device be used on the average, if the new task is admitted? We name this quantity U. knowing U, the total task time and the device's power (i.e., energy spent per unit time) we can calculate AEL. Since U is calculated in the average sense, we are really calculating E[AEL]. The calculation of U is not very simple. It depends on the number of fragments/blocks comprising p and q. Let us first consider the two extremes of the problem. Infinite-block and one-block p and q.

If p and q are infinitely fragmented (with fragments randomly placed), they will occupy the whole [0...1] interval with uniform density. Any sub-interval will have the same properties. Specifically (1-p)·interval of any interval is free and we add q·free_interval if the new task is admitted. Thus the analytical solution is given by equation 1. $U_\infty = (1-p)\cdot q$ eq.(1) The same solution is derived if we considered only one of p or q to be infinitely fragmented. Analytical solutions exist for the one-block and n-block cases too. Due to lack of space they cannot be given here but the interested reader can refer to [2].

The practical question is how does the U_i differ from the U_∞. A full report is given in [2]; here we just summarize the results: For increasing i the difference is decreasing. For i=1 the maximum difference is 0.07 and for i=5 the maximum difference is 0.01. Generally we expect a large number of fragments for p and q, so the U_∞ becomes an excellent estimate having the added advantage of easy computation.

3.2. Real-condintion measurement of energy costs

Being able to calculate the U and the AEL for specific devices, and generally the total energy cost of a specific task based on $\{e_{attr}\}$ and $\mathcal{N}$, is important during admission but is not enough once a task is being admitted. This is because: 1)$\{e_{attr}\}$ are estimations of the devices' usage (the task behavior is influenced also by local state), and 2) even if we had an accurate view of the devices' usage, sharable devices make the computation of AEL in the average sense Once the task is being admitted we have more data on its behavior and we would like to exploit them in order to acquire more accurate energy cost measurements.

It would be useful to list what we need and why we need it. First we need to know the accurate usage of each device by all the admitted tasks cumulatively. This information will be used in the admission control to derive the energy cost of an examined task. Essentially, this information is the $\mathcal{N}$ we are requiring to help us compute the energy cost of a new task. Second we need to know the accurate AEL of any subset of the admitted tasks with respect to the rest of the admitted tasks. In other words: if we were to reject any subset of tasks, how much energy we would have saved? The ability to calculate this quantity is needed while policing the admitted tasks. To calculate all the above quantities each device needs to keep a usage profile for each admitted task (i.e., when, and for how long was a time-sharing device used by task i). To keep such profiles we need support by the devices. The framework in [4] makes provisions for usage profiles to be kept.

4. Admission Control and Policing

Thus far we demonstrated how to derive the node-level energy cost (henceforth denoted as s) and the node-level reward (denoted as r) for a task that wishes to execute in a particular node of remaining energy n. The node has an overall value $V_{\mathcal{P}}(t)$ according to its admission policy $\mathcal{P}$ and the task requests received up to time t. If a task is admitted, reward r is added to the overall value of the node. If a task is rejected, while the remaining energy is enough to accommodate it (i.e., n>=s) then r is subtracted from the overall value as a penalty. If there is not enough energy to accommodate a task, $V_{\mathcal{P}}$ is left unchanged. An admission control policy $\mathcal{P}$ specifies if a task is admitted given r, s, and n. That is, $\mathcal{P}(r, s, n) \in \{0, 1\}$, 0 denoting rejection and 1 denoting acceptance.

The problem of admission control is to specify the optimal $\mathcal{P}$ so as the $V_{\mathcal{P}}$ accumulated until the remaining energy is zero, is maximized. This problem is reminiscent of the well-known knapsack problem. In our case though, the items that undergo admission (i.e., the tasks) are not all known a priori but instead they are coming as requests, as time passes. Furthermore, their rewards r, and costs s, are not fixed but are distributed following a joint distribution function F_{rs}. This enhanced version of the knapsack problem is called Dynamic and Stochastic Knapsack Problem (DSKP), and it was previously studied in operations research [7][10]. In [7] an optimal admission policy for DSKP is discovered. Inspired by the solution in [7] we solved our specific problem, obtaining the desired optimal policy. The solution given here is self-contained and does not require any knowledge from [7]. The reference is only made to acknowledge our initial inspiration for the solution.

Imagine a node with remaining energy n. We define the quantity V(n) as the *expected* overall value $V_{\mathcal{P}}$ the node accumulates until its remaining energy is zero, under some policy $\mathcal{P}$. V(n) for any policy is given by the recursive formula in equation 2.

V(n) = P(task i accepted) · { E[r_i | task i accepted] + E[V(n-s_i) | task i accepted] } + P(task i rejected) · V(n) - P(task i rejected and n>= s_i) · E[r_i | task i rejected and n>= s_i] eq.(2)

E[] denotes average value, P() denotes probability, and | denotes "given that". The formula simply states that if a task *i* is accepted the new value is $r_i + V(n\text{-}s_i)$ (i.e., reward plus the expected value of the remaining energy), if a task is rejected we still have the expected value of our current energy, and if the rejection was <u>not</u> forced (i.e., $s_i \leq n$) then r_i is subtracted form the expected value.

If we assume that r and s take discrete values and we know their joint distribution function F_{rs}, equation 2 is transformed to equation 3. We can also assume continuous values and just replace the sums with integrals and the joint distribution function with a joint probability density function in equation 3. We chose discrete values to facilitate numerical computations.

$$V(n) = \sum_{r_i,s_i:\,task\ \ i\ accepted} r_i \cdot F_{rs}(r_i,s_i) + \sum_{r_i,s_i:\,task\ \ i\ accepted} V(n-s_i)\cdot F_{rs}(r_i,s_i) + \sum_{r_i,s_i:\,task\ \ i\ rejected} V(n)\cdot F_{rs}(r_i,s_i) - \sum_{r_i,s_i:\,task\ \ i\ rejected\ \ and\ \ n\geq s_i} r_i \cdot F_{rs}(r_i,s_i) \quad \text{eq.(3)}$$

Solving for V(n) we get equation 4.

$$V(n) = \frac{\sum_{r_i,s_i:\,taski\ accepted} r_i \cdot F_{rs}(r_i,s_i) + \sum_{r_i,s_i:\,taski\ accepted} V(n-s_i)\cdot F_{rs}(r_i,s_i) - \sum_{r_i,s_i:\,taski\ rejected\ \ and\ \ n\geq s_i} r_i \cdot F_{rs}(r_i,s_i)}{(1 \ - \sum_{r_i,s_i:\,task\,i\,rejected} F_{rs}(r_i,s_i))}$$

eq.(4)

If an admission policy does not depend on V(n), then the summation indexes do not depend on V(n). In such a case we can easily calculate equation 6 recursively, starting with V(0) = 0. Two examples of policies that do not depend on V(n) are: 1)accept all tasks (i.e., no policy), and 2)accept tasks if r/s >= threshold. If a policy does depend on V(n) (e.g., the optimal policy, as shown later) then it is harder to compute V(n). Fortunately, for the optimal policy we are able to find an iterative algorithm that converges fast and computes V(n).

Let us see now how is the optimal admission control policy defined. Assume that we do follow the optimal policy so that we have an expected value $V_{opt}(n)$ for remaining energy n. If a task arrives at this point and we are indeed faced with a decision (i.e., n>=s) we can calculate the value for both outcomes of the decision. If the task is accepted the overall value will become $r + V_{opt}(n\text{-}s)$. If the task is rejected the overall value will become $V_{opt}(n) - r$. The decision is evident now: If $r + V_{opt}(n\text{-}s) >= V_{opt}(n) - r$ we admit the task otherwise we reject it. This gives us the optimal policy.

$$\mathcal{P}_{\text{opt}}(\text{r, s, n})=\begin{cases}1 \quad if\ r \geq \dfrac{V_{opt}(n)-V_{opt}(n-s)}{2} \quad AND \quad s \leq n \\ 0 \quad if\ r < \dfrac{V_{opt}(n)-V_{opt}(n-s)}{2} \quad OR \quad s > n\end{cases}$$

eq.(5)

Going back to equation 4, we observe that with the optimal policy the summation indexes are affected by V(n). So equation 4, apart from being a recursive formula, it now becomes a highly non-linear equation of V(n). As stated earlier though, we constructed an iterative algorithm that converges fast towards V(n). The details of the algorithm and the convergence properties are beyond the scope of this paper. The interested reader can refer to [3].

Figures 1, 2, 3, and 4, are plotting V(n) versus n for a variety of policies (including the optimal) and for four joint distribution functions F_{rs}. Rewards take values in the interval [0.1...5] with granularity 0.1. Costs take values in the interval [1...50] with granularity 1.For the first distribution function, r and s are independent and uniform. For the rest of the distribution functions, r and s are correlated following different types of correlation. If we know (or can estimate) all the possible tasks along with their energy costs, their rewards, and their occurrence probabilities, then we can compute F_{rs}. If this information is not available we can still perform quite well with "blind" policies, as we will see by the analysis of figures 1-4.

From figures 1-4 we observe that with the optimal admission policy we earn up to 48% more in V(n) (48% achieved for the F_{rs} in Figure 4) than with no policy (i.e., accept all tasks i, given that $s_i \leq n$). With uniform F_{rs} the optimal policy achieve a 27% increase in V(n). For the F_{rs} in Figure 2 (where it is more probable to have large r with large s and vice versa) the optimal policy offers only a 5% increase of V(n). The figures also show the V(n) achieved by "ratio threshold" policies. We see that policies of some thresholds follow the optimum V(n) very closely, while others perform worse than the absence of policy, or even have negative slopes (e.g., the 0.19 threshold policy in Figure 4.

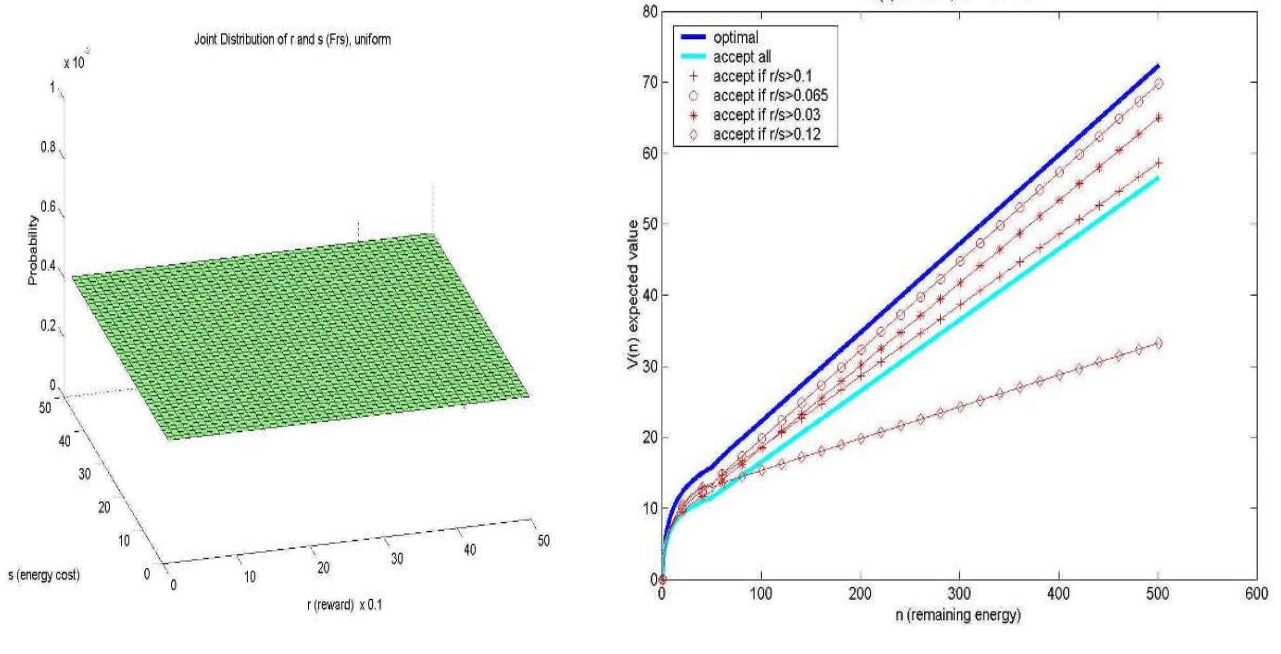


Figure 1: V(n) for various policies and uniform F_{rs}

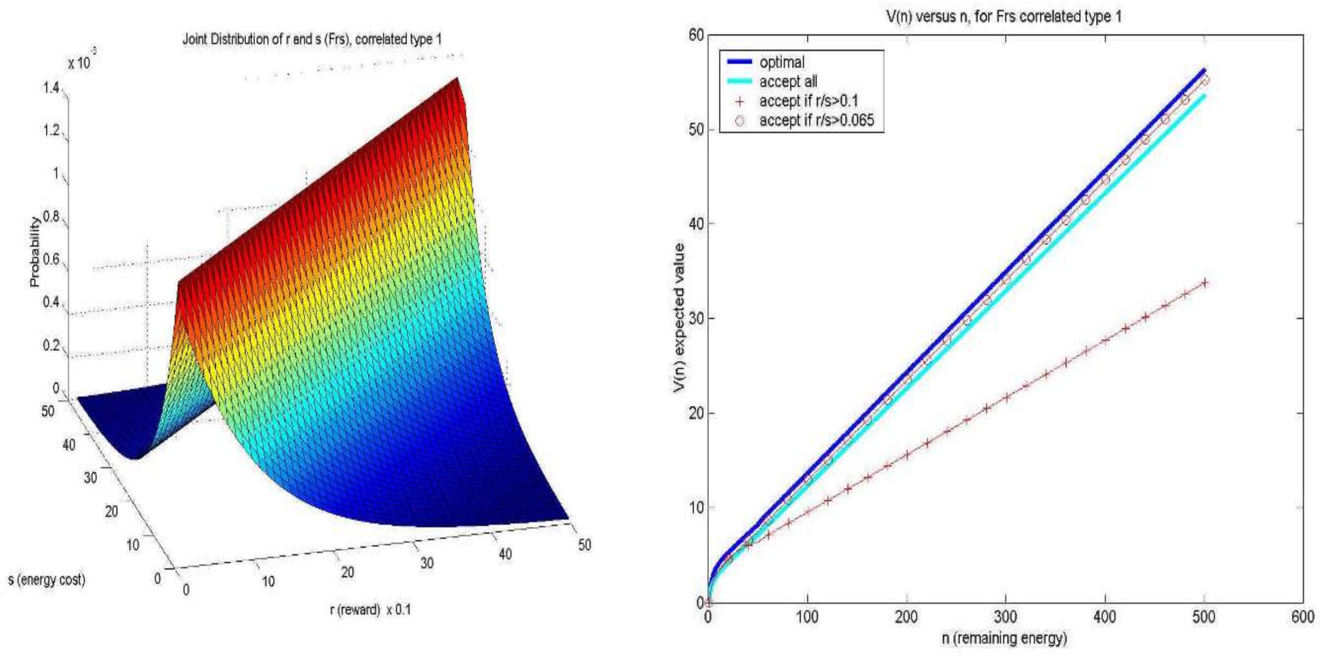


Figure 2: V(n) for various policies and F_{rs} (type1)

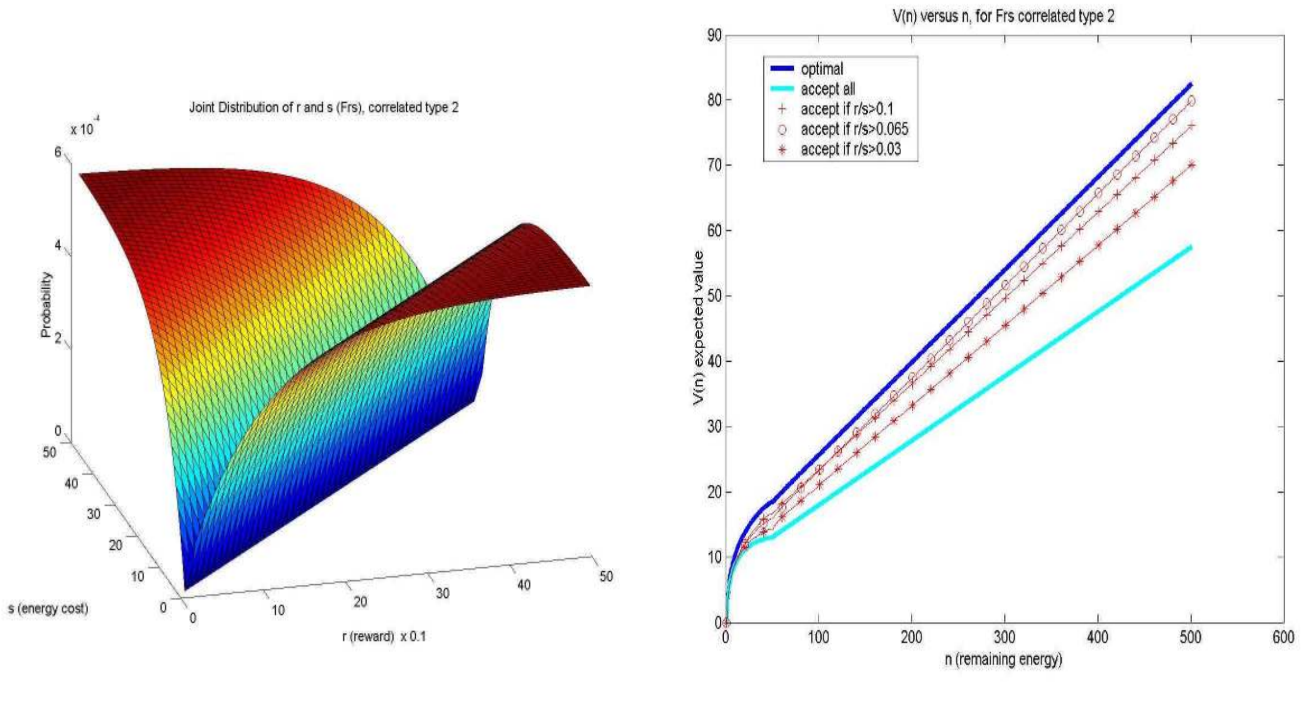
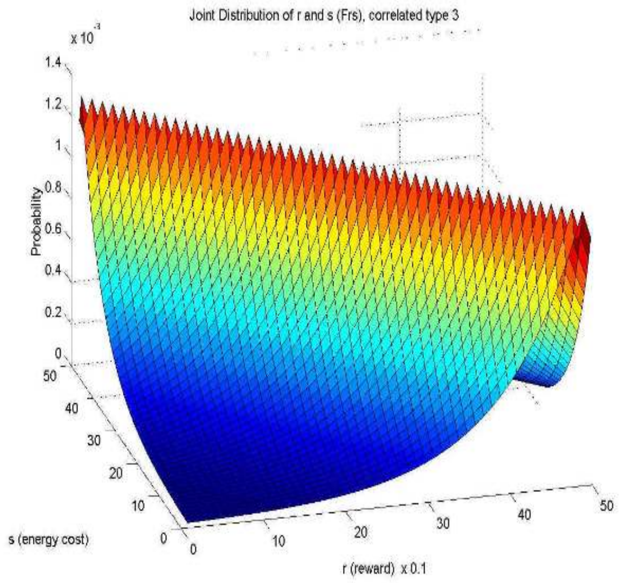


Figure 3: V(n) for various policies and F_{rs} (type2)

0-7695-1893-1/03 $17.00 © 2003 IEEE



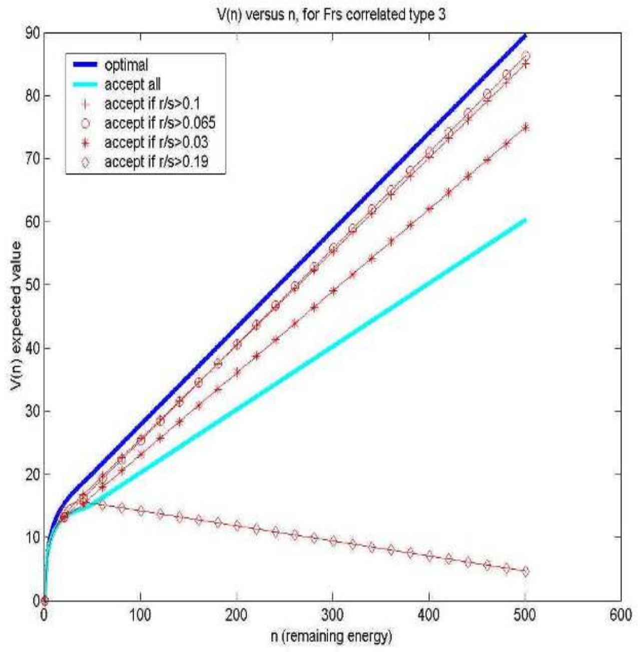


Figure 4: V(n) for various policies and F_{rs} (type3)

The interesting part though is the properties of the optimal policy. From the figures we see a strong indication that $V_{opt}(n)$ is linear with respect to n, $\forall$ $n>n_{linear}$. In particular, n_{linear} seems to be equal to max(s)=50 from observing the figures. Trying to find n_{linear} by finding the point that the second derivative of $V_{opt}(n)$ becomes zero, reveals the value n_{linear} =2max(s)=100. We should not forget though that these are numerically computed data with finite accuracy, thus it might also be the case that $V_{opt}(n)$ is converging to be linear with respect to n, (or stated otherwise the second derivative of $V_{opt}(n)$ is converging to zero).

If we show that $V_{opt}(n)$ is linear or converges to linear we can prove some interesting properties. Assume that $V(n) = a\cdot n + b,\ \forall n>n_{linear}$. Then $V(n)-V(n-s) = a\cdot s,\ \forall s$ *and* $\forall\ n>n_{linear}+max(s)$. Thus if $V_{opt}(n)$ is linear with respect to n then the difference $V_{opt}(n)$-$V_{opt}(n-s)$ is linear with respect to s. Going back to equation 5 and substitute $V_{opt}(n)$-$V_{opt}(n-s)$ with $a\cdot s$ we get:

$$\mathscr{P}_{opt}(r, s, n) = \begin{cases} 1 & if\ \dfrac{r}{s} \geq \dfrac{a}{2} \quad AND \quad s \leq n \\ 0 & if\ \dfrac{r}{s} < \dfrac{a}{2} \quad OR \quad s > n \end{cases} \qquad \text{eq.(6)}$$

Equation 6 states that if $V_{opt}(n) = a\cdot n + b,\ \forall n>n_{linear}$ **then the optimal policy is a "ratio threshold" policy with threshold a/2 $\forall\ n>n_{linear}+max(s)$.** This observation explains the very good performance of some ratio threshold policies in the figures. If we choose the right threshold (i.e. close to a/2) then the resulting V(n) will closely follow $V_{opt}(n)$ trailing only by a constant offset. The offset is created at the non linear region of V(n) (i.e., 0<n<max(s)) where the optimal policy does not keep a constant ratio threshold. Another very important point to notice is that the optimal ratio a/2 does not change considerably for radically different F_{rs}. The ratio of 0.065 performs extremely well for all F_{rs} tested. Thus even if F_{rs} is unknown and we just know the max(r) and max(s), we can still achieve good performance by computing a/2 for one random F_{rs} (e.g. the uniform). If F_{rs} is known, the computation and storage needed to enforce the optimal admission policy is minimal. Concerning storage, we need to keep the values of V(n) for the non-linear region (i.e., 0<n<max(s)) and the optimal ratio a/2. Concerning computation, we need to perform a division (r/s) and a comparison with a constant [a/2] for the linear region, or a comparison with the result of a subtraction [V(n)-V(n-s)] for the non linear region.

Finally, we wish to address the topic of policing. With the mechanisms described in section 3.2, our energy management system measures the energy cost of each admitted task with respect to the rest of the admitted tasks every p_{attr1} units of time. This measurement acts as an estimation for the *future remaining energy cost* of each admitted task (named s'_i). This estimation is more accurate than the initial *mean* energy cost used by the admission policy. Thus, at any time t, we have an estimation of the remaining energy cost s'_i, and the remaining reward r'_i task i can offer (based on the initial r_i, p_{attr2}, and the total task execution time). Obviously this poses a policing problem.

Do we evict any tasks? When? Which ones? The "when" question is answered easily. The check is performed every time there is a change to the estimated energy cost or to the remaining reward of any task. The optimal policing strategy would be to get all possible subsets of the admitted tasks and consider them for eviction. For each (possibly) evicted subset compute the quantity: V_{pol}(tasks evicted) = $\sum_{i:\,task\ i\ kept} r'_i - \sum_{i:\,task\ i\ evicted} r'_i + V_{opt}(\ n - S(tasks\ kept)\ )$, where n is the remaining node energy and *S(set_of_tasks)* is the estimation of the cumulative energy spend by the *set_of_tasks* till their completion. Naturally, *S()* is not simply the sum of individual costs since we have sharable devices.

The subset that maximizes V_{pol} is the one that should be evicted (note that the empty set is a valid choice in this procedure). Given the non-linearity of function S we have to resort to brute force and simply check all possible subsets. For a total of k tasks currently admitted, there are $\sum_{l:\,0..k} \frac{k!}{(k-l)!\,l!}$ possible subsets. Indicatively, for k= 5 the possible subsets are 32, and for k=10 they rise to 924. This might be a large number to check every time there is a change in one s'_i or r'_i. Thus we propose a heuristic for policing. Each time there is a change in one s'_i or r'_i we only check V_{pol} for the two extreme subsets (i.e., the "empty set", and "all the currently admitted tasks"). If the V_{pol} for keeping all the tasks is larger than the V_{pol} for evicting all the tasks, do nothing. Otherwise, run the brute force algorithm and find the best subset of tasks to evict. The rationale behind this heuristic is that most of the time the tasks are well behaved and stay within their pre-admission cost and reward ratio, so no task needs to be evicted. Our immediate plans include the evaluation of this heuristic and generally the exploration of the trade-off between V_{pol} achieved vs. computation made.

5. Conclusions

In this paper we are concerned with the problem of managing the finite energy in a WASN. Given a finite energy amount and a sequence of application requests (chosen among a set of probable applications) with different energy costs and different rewards for the user, the question is to apply an admission control policy so as to maximize user rewards. Although the problem would be optimally solved at the system level, a mechanism that adopts such a viewpoint will require from the distributed applications to have full knowledge of each other or to exchange a large number of messages due to the large solution space. In either case a system-level solution becomes impractical for a WASN that hosts multiple, often transient users. Instead, we propose a mechanism that solves the energy management problem at the node level without requiring for the applications to directly communicate with each other. The application's system-level behavior (i.e., distribution in nodes, choice of algorithm incorporated) is still application-specific but now it is decided by using the acceptance/rejection replies received from individual nodes, and not by direct negotiations with the other applications. Solving the problem at the node level first requires a proper formulation. The formulation reveals some sub-problems, such as calculating the Added Energy Load that a task is bringing to set of tasks already running in the node, and measuring the devices' usage profiles in real time. After these sub-problems are solved, the optimal admission policy is presented and analyzed. From the study of the numerical data we can see the gains one can receive by applying the optimal policy compared to the absence of an admission policy. Finally, we show that the optimal policy can easily be applied using the limited computation powers of a node, if we know the distributions of rewards and energy costs. Most importantly though, the optimal policy can be closely approximated by a minimally computational intensive "ratio threshold" policy, even in the absence of distribution information.

6. References

[1] R. Amirtharajah and A.P. Chandrakasan, "Self-powered signal processing using vibration-based power generation," IEEE Journal of Solid State Circuits, vol.33, no.5 May, 1998.

[2] A. Boulis, "Calculation of AEL for sharable devices", TM-UCLA-NESL-2003-01-001, http://nesl.ee.ucla.edu/TMs.

[3] A. Boulis, "Method for numerically solving the optimum policy equation for admission control in WASNs", TM-UCLA-NESL-2003-01-002, http://nesl.ee.ucla.edu/TMs.

[4] A. Boulis, C. C. Han, and M. B. Srivastava, " Design and Implementation of a Framework for Efficient and Programmable Sensor Networks", To appear in proccedings of MobiSys 2003, San Fransisco, CA, May 5-8, 2003.

[5] A. Chandrakasan, R. Amirtharajah, S.H. Cho, J. Goodman, G. Konduri, J. Kulik, W. Rabiner, and A. Wang "Design Considerations for Distributed Microsensor Systems," Proc. IEEE 1999 Custom Integrated Circuits Conference (CICC '99), May 1999, pp. 279-286

[6] C. Jaikaeo, C Srisathapornphat, and C. Shen, "Querying and Tasking of Sensor Networks", SPIE's 14th Annual International Symposium on Aerospace/Defense Sensing, Simulation, and Control (Digitization of the Battlespace V), Orlando, Florida, April 26-27, 2000.

[7] A.J. Kleywegt and J.D. Papastavrou, "The Dynamic and Stochastic Knapsack Problem", Operations Research, 46, pp. 17-35, 1998.

[8] John Kymisis, Clyde Kendall, Joseph Paradiso, and Neil Gershenfeld, "Parasitic Power Harvesting in Shoes," Second IEEE International Conference on Wearable Computing (ISWC), Pittsburgh, PA, October 1998.

[9] P. Levis, D. Culler, "Maté: A Tiny Virtual Machine for Sensor Networks." Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X), October 5-9 2002.

[10] A. Marchetti-Spaccamela and C. Vercellis. "Stochastic on-line knapsack problems", Mathematical Programming, 68(1):73–104, Jan 1995.