

Energy-aware Scheduling of Real-Time Tasks in Wireless Networked Embedded Systems

G. Sudha Anil Kumar, G. Manimaran and Z. Wang
Real-Time Computing and Networking Laboratory
Dept. of Electrical and Computer Engineering
Iowa State University, Ames, IA 50011
Email: {anil,gmani,zhengdao}@iastate.edu

Abstract

Recent technological advances have opened up several distributed real-time applications involving battery-driven embedded devices with local processing and wireless communication capabilities. Energy management is the key issue in the design and operation of such systems. In this paper, we consider a single-hop networked real-time embedded system where each node supports both dynamic voltage scaling (DVS) and dynamic modulation scaling (DMS) power management techniques to tradeoff time for energy savings. In this model, we address the problem of scheduling periodic complex tasks where each task consists of several precedence constrained message passing sub-tasks. Our contributions towards this problem are two fold. First, we analyze the system level energy-time tradeoffs considering both the computation and communication workloads by defining a novel energy gain metric. We then present static (centralized) and dynamic (distributed) energy gain based slack allocation algorithms which reduce the total energy consumption, while guaranteeing the ready time, deadline and precedence constraints. We compare the performance of the proposed algorithms with several baseline algorithms through simulation studies. Our results show that the proposed algorithms perform significantly better than the baseline algorithms for the simulated conditions. Finally, we identify several interesting energy-aware research problems in the area of networked real-time embedded systems.

1 Introduction

Real-time systems have undergone an evolution in the last few years in terms of their number and variety of applications as well as complexity. A natural result of these advances coupled with those in sensor techniques and networking have led to the rise of a new class of application

which fall into the distributed real-time embedded systems category [1]. Recent technological advancements in device scaling have been instrumental in enabling mass production of such devices at reduced costs. As a result, applications with a number of inter-networked embedded devices have become prominent. Extensive research has already been carried out to achieve real-time guarantees over a set of nodes distributed over wired networks [2]. However, there exists a number of real-time applications in domains such as industrial processing, military, robotics and tracking, which require the self-powered nodes to communicate over the wireless medium where application dynamics prevent the existence of a wired infrastructure. These applications present challenges beyond those of traditional embedded or networked systems as they aim to provide critical real-time guarantees over a severely resource-constrained network with battery-driven nodes.

A typical architecture in a networked real-time embedded system consists of several processor controlled nodes interconnected via the wireless network. The system software running on each node enables the execution of one or more concurrent tasks which are activated by the arrival of triggering events generated by the external environment, a timer or arrival of a message from another task. A response to an event generally involves several tasks to be executed on different nodes and several messages to be exchanged in the network. For the proper functioning of the whole system, each individual task as well as all the messages exchanged need to complete before stringent deadlines while incurring as less energy as possible. An effective energy-aware strategy for such a system would appropriately leverage the low power modes supported by different components within each node of the network under a unified system-level perspective.

Majority of the existing research work in the area of real-time energy-aware systems focused on reducing the energy consumption of the processor employing the well known

dynamic voltage scaling (DVS) technique [7]. DVS refers to the technique of simultaneously varying the processor voltage and frequency as per the performance level required by the tasks. DVS allows to trade off processor’s speed for energy savings. Several energy aware real-time DVS algorithms have been proposed addressing a wide range of task scheduling problems for a variety of system models [7, 6].

Recently, the focus of the research community has shifted from processor level energy management to system-level energy management, wherein the objective is to minimize the total energy consumption of the entire system as opposed to minimizing the processor energy consumption alone. More specifically, the interplay between the DVS technique employed to reduce the processor energy consumption and the rest of the system is being actively studied. In [8], an optimal processor frequency has been analytically derived to minimize the system energy consumption considering both the on-chip and off-chip workloads. The proposed solutions are effective for computation intensive real-time applications and do not address the communication aspects of the system.

The communication energy consumption of the system is considered in [9]. In their system model, the authors consider a DVS enabled processor along with a wireless card that independently employs a dynamic power management strategy wherein the card transitions between an active and several low power states as per the out-going traffic. Depending on the current state of the network card, the processor frequency is varied to control the finish times of the tasks which are same as the release times of the corresponding messages. This is done with the aim of maximizing the sleep durations for the network card and minimizing the sleep-to-active and active-to-sleep transition overheads. This work however does not consider message deadlines making it unsuitable for real-time applications where timely message delivery is of paramount importance.

From a different perspective, more effective power management techniques [5] can be employed to reduce the communication energy consumption as compared to low-power sleep modes that are assumed in the above work. Some such techniques include, power adaptation where transmission power is adapted based on the source-destination distance and dynamic modulation scaling (DMS) where the modulation level of the out-going message can be reduced to achieve transmission energy reductions [4].

Similar to the DVS technique employed in processor energy management, DMS allows to tradeoff message transmission times for energy savings. The dynamic changes in the modulation level of the outgoing packets are communicated to the destination via the packet header and hence, the overheads of modulation scaling are almost negligible. Several energy aware DMS based schemes have been developed for non-real-time network applications [3] which

cannot be directly extended to handle real-time workloads. In [4], the energy-aware problem of scheduling real-time messages at a single node is considered and a DMS based scheme has been presented. However, presented scheme does not consider the local computation issues and works at a node level ignoring the workload in the rest of the network. To the best of our knowledge, ours is the first work which addresses the system-level energy management problem considering both the computation and communication workloads in the network with real-time constraints.

2 System Model

We consider a single-hop wireless networked embedded system with n nodes which share the common wireless medium that is accessed in an exclusive manner. We assume all nodes in the network are time synchronized. Each node supports DVS with k_t discrete frequency levels and DMS with k_m discrete modulation levels. In the examples and simulation studies we use the PXA255 processor’s power specifications [10] shown in figure 1.(b) to model the processor energy consumption.

Task Model: We consider m periodic complex real-time tasks whose deadline is same as the period. Each complex task consists of several message exchanging sub-tasks where each sub-task has precedence constraints with other sub-tasks. For simplicity, in the rest of the paper we refer to sub-tasks as tasks. For each given complex task, all its tasks and messages need to complete their execution before the deadline, respecting the precedence constraints. Figure 1.(a) shows the precedence graph of an example complex real-time task where nodes denote the tasks and edges denote the messages. We use the following notation in the rest of the paper. For each task T_i , $desc(T_i)$ denotes the set of messages which are produced by it. For example, $desc(T_2) = \{M_1, M_2\}$. Similarly, $pred(T_i)$ denotes the predecessor messages of T_i . For example, $pred(T_5) = \{M_3, M_4\}$. The workload for the tasks is specified as the number of CPU cycles (denoted as CC) and for the messages it is specified as the number of bits (denoted as L). We assume tasks and messages are non-preemptible.

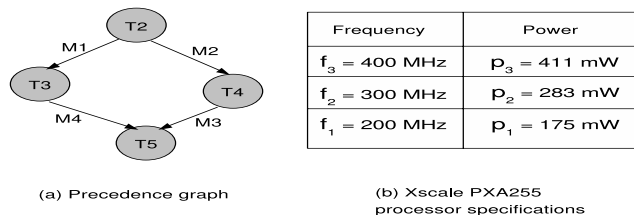


Figure 1.

The following issues need to be handled while schedul-

ing such complex tasks in a networked embedded system with several embedded devices: (1) Task allocation - mapping individual tasks to nodes, (2) Scheduling - scheduling local tasks at each node and messages in the shared wireless medium, (3) Assigning frequency and modulation levels to the individual tasks and messages, respectively. Rich literature exists addressing the first two issues [2]. Therefore, in this paper we assume that task allocation and scheduling have been performed a priori to obtain a feasible schedule. *Our primary focus is on assigning frequency and modulation levels to different tasks and messages respectively, to reduce the total energy consumption of the input schedule. This involves performing slack allocation across tasks and messages without violating the deadline and precedence constraints.*

In the rest of the paper, we use the generic term *performance level* whenever we intend to refer to either a task's frequency level or a message's modulation level. Similarly, we use the generic term *entity* to refer to either a task or a message in the schedule. In these terms, our goal is to assign appropriate performance levels to different entities in the input schedule with the objective of minimizing the total energy consumption while not violating the deadline and precedence guarantees provided by the input schedule.

Communication Model: We assume that the modulation is Quadrature Amplitude Modulation (QAM). The channels are modeled as frequency-flat Rayleigh fading. Let b denote the number of bits per modulation constellation symbol. The constellation size is denoted as $M = 2^b$. Let E_b denote the received energy per bit, $N_0/2$ denote the channel noise power spectral density, E_s denote the received energy per constellation symbol, d_{min}^2 the minimum average squared Euclidean distance between two constellation symbol at the receiver, and BER denote the bit error rate. We have the following relationships:

$$b = \log_2 M \quad (1)$$

$$BER \approx N_0/d_{min}^2 \quad (2)$$

$$d_{min}^2 = \frac{6}{M-1} E_s \quad (3)$$

$$E_s = bE_b \quad (4)$$

The approximation in (2) is valid for high signal-to-noise ratio (SNR). Combining these, we have

$$E_b \approx \frac{2^b - 1}{6b} \cdot \frac{N_0}{BER}. \quad (5)$$

If a message contains L bits, then the total necessary received energy will be $E_L = L \cdot E_b$. We assume that the propagation loss follows a polynomial model: the power decays in the α -th order of the distance:

$$E_{ts} = \left(\frac{d}{d_0}\right)^\alpha E_L \quad (6)$$

where E_{ts} is the necessary transmitted energy to achieve a received energy of E_L , d is the distance between the transmitter and the receiver, d_0 is a normalizing constant that depends on the wavelength. Usually α is between 2 and 5. As a result, the total transmitted radio energy can be expressed as

$$E_{ts} = \left(\frac{d}{d_0}\right)^\alpha \frac{L(2^b - 1)}{6b} \cdot \frac{N_0}{BER} \quad (7)$$

In addition to the E_{ts} , some energy is consumed by the circuitry during the transmission which is given by: $E_{tc} = \frac{LC_t}{\log_2 M}$. Similarly, the energy consumed in receiving a message is given by: $E_{rc} = \frac{LC_r}{\log_2 M}$. Here C_t and C_r are implementation dependent constants[4]. In the rest of the paper, we assume $N_0 = 4 * 10^{-13}$, $BER = 10^{-6}$, $C_t = 75nJ$, $C_r = 100nJ$, $L = 1024$ and $W = 1KHz$ as the default values.

The time taken for transmitting a L -bit message in the shared wireless medium is given by:

$$T = \frac{L}{Wb} \quad (8)$$

Here, W denotes the bandwidth (symbols per second) of the channel in hertz and the bandwidth in bits per second can be calculated as Wb . We assume each node supports DMS technique where modulation level (b) can be varied to tradeoff transmission latency for energy savings. Further, we assume that the BER is designed low enough to provide the necessary message reliabilities.

Notations: In the rest of the paper, we use the following notation for each entity e_i in the schedule.

- $R(e_i)$: Ready time of the complex task to which e_i belongs to.
- $D(e_i)$: Deadline of the complex task to which e_i belongs to.
- $st(e_i)$: start time of e_i in the schedule.
- $ft(e_i)$: finish time of e_i in the schedule.
- $T(e_i, p_\alpha)$: processing time of entity e_i when operated at the performance level, p_α . For a task, it is calculated as $\frac{C}{p_\alpha}$ where p_α refers to the operating frequency. Similarly, for a message it is calculated as $\frac{L}{Wp_\alpha}$ where p_α is its modulation level.
- $E(e_i, p_\alpha)$: energy consumption of entity e_i when operated at the performance level, p_α .
- $est(e_i)$: denotes earliest start time by which e_i can be scheduled without violating any of the constraints. It is calculated as, $est(e_i) = \text{Max}\{prev_ft(e_i), pred_ft(e_i), R(e_i)\}$. The expressions $prev_ft(e_i)$ and $pred_ft(e_i)$ are defined below.
- $lst(e_i)$: denotes latest start time at which e_i can be scheduled without violating any of the constraints. It is calculated as, $lst(e_i) = \text{Min}\{next_st(e_i), desc_st(e_i), D(e_i)\} - T(e_i, p_\alpha)$. The expressions $next_st(e_i)$ and $desc_st(e_i)$ are defined below.

- $host(e_i)$: denotes the processor on which e_i is scheduled. If e_i is a message, then it refers to the common shared wireless medium. In the schedule shown in figure 3, $host(T_2) = host(T_1) = P_2$ and $host(M_2) = host(M_1) = channel$.
- $prev(e_i)$: entity which is schedule right before e_i on the same processor or channel. In the schedule shown in figure 3, $prev(T_2) = T_1$ and $prev(M_2) = M_1$.
- $next(e_i)$: entity which is schedule right after e_i on the same processor or channel. In the schedule shown in figure 3, $next(T_1) = T_2$ and $next(M_1) = M_2$.
- $pred_ft(e_i)$: earliest time by the which all the predecessor entities of e_i will finish. Mathematically, $pred_ft(e_i) = Max_{\forall M_j \in pred(e_i)} \{est(e_j) + T(e_j, p_\alpha)\}$.
- $prev_ft(e_i)$: earliest time by which the $prev(e_i)$ completes and is calculated as $est(prev(e_i)) + T(prev(e_i), p_\alpha)$.
- $desc_st(e_i)$: the latest time by which at least one of the descendant entities of e_i will start in the schedule. It is calculated as, $desc_st(e_i) = Min_{\forall e_j \in desc(e_i)} \{lst(e_j)\}$.
- $next_st(e_i)$: the latest time by which $next(e_i)$ will start in the schedule.

3 Energy-aware scheduling Algorithms

In this section, first we analyze the system-level energy-time tradeoffs by defining a novel metric called normalized energy gain. We then present an offline static scheduling algorithm followed by a dynamic distributed scheduling algorithm.

3.1 System level energy-time tradeoffs

In order to effectively reduce the total energy consumption of the input schedule the available slack should be allocated across tasks and messages in the schedule. A good slack allocation strategy would allocate slack to the entity which results in maximum energy reduction for every additional unit of slack allocated to it. To determine the best entity for slack allocation, we define the following metric called *normalized energy gain* for each entity as follows:

$$G(i, i-1) = \frac{E_i - E_{i-1}}{T_{i-1} - T_i} \quad (9)$$

Where E_i and T_i respectively denote the energy consumption and time incurred by the entity when operated at i^{th} performance level. For each entity e_j which is currently assigned the i^{th} performance level, $G(i, i-1)$ represents the energy reduction that would be obtained by reducing its performance level to $i-1$ normalized with respect to

the additional time incurred for operating it at performance level $i-1$ instead of level i . The energy gain metric succinctly captures how effectively a given amount of slack is utilized by each entity and ideally slack should be allocated to the entity which offers highest normalized energy gain.

Figure 2 shows the normalized energy gain for the mes-

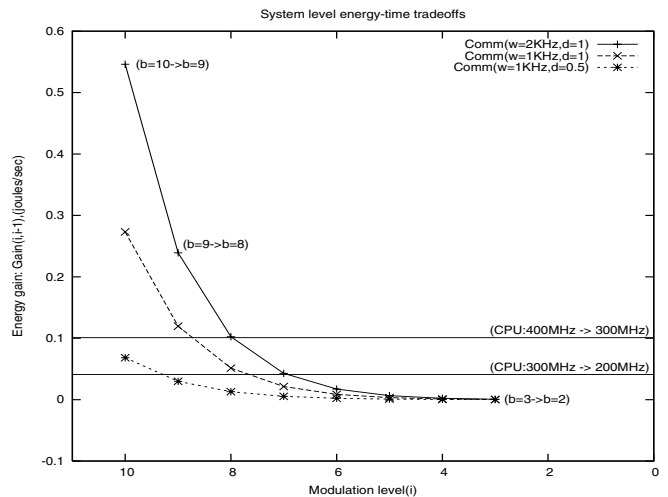


Figure 2. System level energy-time tradeoffs

sages as a function of i (modulation level) for different values of W and d with message size, $L = 1$. It also shows the DVS energy gains (horizontal lines) offered by a task of size $CC = 1$ for the PXA255 processor settings. The following two important observations can be made from the figure.

1. As we decrease the performance level, the subsequent energy gains obtained are decreasing both for the messages and tasks as shown in the figure. For example, the energy gain obtained by reducing the CPU frequency from $300MHz$ to $200MHz$ is lower than that obtained by reducing the frequency from $400MHz$ to $300MHz$. Similarly, for a given W and d , we have $G(i+1, i) > G(i, i-1)$. This trend suggests that we should allocate slack incrementally across messages and tasks keeping track of the decreasing energy gain values.
2. The energy consumption of message transmission is in general higher than that of computation and it may appear on the surface that the slack should be entirely allocated to the messages. However, from the figure 2 we can see that the exact energy gains depend upon several parameters like W , d , the current modulation level b of the message and the current frequency level of the task. For example, in the figure considering the curve with $W = 1KHz$ and $d = 1$, we have $G(i=8, i=7) < G(400MHz, 300MHz)$ where as $G(i=9, i=8) > G(400MHz, 300MHz)$. This suggests that there are situations (specific values of W and d) where slack must be allocated to tasks rather

than messages. For the cases, where the message energy gains are higher than the task energy gains, the energy gain metric helps comparing the energy reductions offered by each message and guides the slack allocation across different messages.

In order to ensure a safe slack allocation mechanism, we estimate the maximum safe slack for each entity prior to allocating any slack to it. For a given input schedule several task movement techniques like *sliding*, *passing* and *task migration* can be applied to determine the maximum safe slack for each entity. In this paper, we use the sliding technique due to its simplicity. The proposed algorithms can be easily extended to use the other techniques. In sliding, no entity e_i is allowed to start before $prev(e_i)$ finishes. Further, each entity e_i should complete before its $next(e_i)$ starts execution in the schedule and $host(e_i)$ should not be modified at any point of time. Following this, the maximum available slack for each entity e_i can be calculated as $lst(e_i) - est(e_i) - T(e_i, p_\alpha)$ where p_α is the currently assigned performance level of e_i .

3.2 Gain based Static Scheduling (GSS)

We now present our offline gain based scheduling algorithm which assigns performance levels to each entity in the input schedule. The GSS algorithm proceeds in iterations allocating slack in an incrementally fashion. In each iteration, the highest energy gain yielding entity is chosen and it is allocated just enough slack to reduce its performance mode by one level. The rest of the slack is allocated similarly to the remaining entities in the schedule. At the end of each iteration, the individual energy gain values are updated.

In order to keep track of the messages which can utilize more slack without affecting any constraints, the GSS algorithm maintains a set Q and updates it after each iteration. The algorithm terminates once the set Q becomes empty. The following step by step procedure presents the GSS algorithm.

The worst-case computational complexity of the GSS algorithm can be derived as $O((n+m)(nk_t + mk_m))$ where n and m respectively denote the number of tasks and messages in the schedule.

The illustrative example shown in figures 3 and 4 demonstrates the working of the GSS algorithm. Figure 3 represents the input schedule. We are primarily interested in tasks T_2, T_3, T_4, T_5 and messages M_1, M_2, M_3, M_4 whose precedence relationship is shown in figure 1.(a). For simplicity, we assume the other tasks in the schedule are independent; further, we assume a unit normalized source-destination distance for all the messages in the schedule. Our algorithms however work for the general cases.

The GSS algorithm will first pickup the highest energy gain yielding entity, M_4 which offers an energy gain of

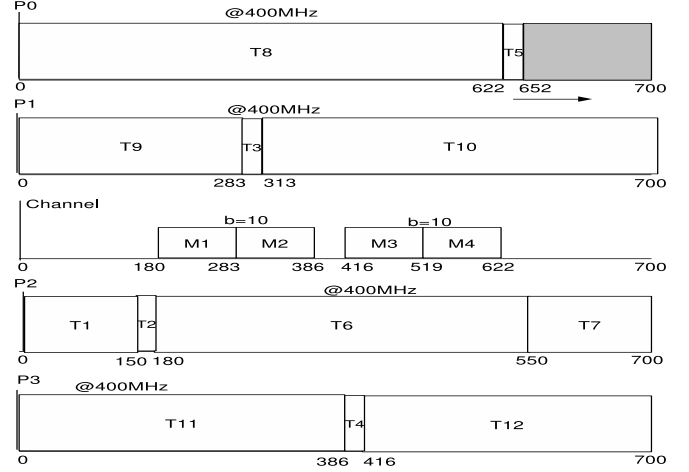


Figure 3. Input Schedule

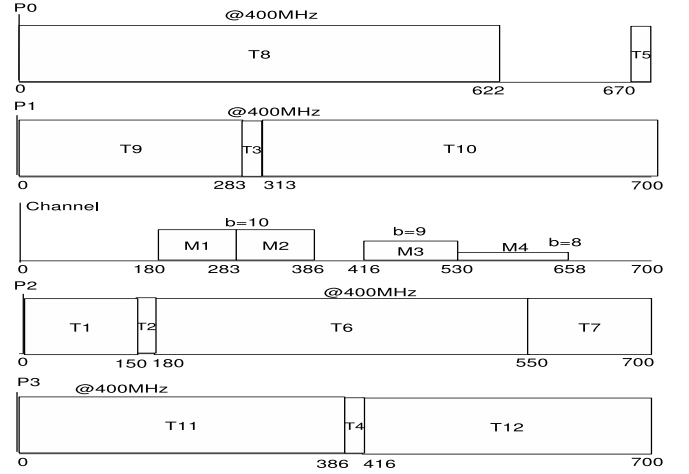


Figure 4. Working of the GSS algorithm

$3107uJ (= \frac{N_0}{6BER} L(\frac{2^{10}-1}{10} - \frac{2^9-1}{9}))$ using $11ms$ (obtained as $\frac{L}{9W} - \frac{L}{10W}$) of slack. In fact, M_1, M_2, M_3 and M_4 have equal energy gains (i.e, $3107/11 uJ/ms$) at this point. However M_1 and M_2 have zero maximum slack and the tie between M_3 and M_4 is arbitrarily broken. Now, M_4 's maximum available slack is determined by sliding T_5 towards the right as shown in the figures. Since $M_4 \in pred(T_5)$, it will now have $48ms$ of slack to lower its modulation level. Hence, the GSS algorithm reduces the modulation level of M_4 to $b = 9$ using $11ms$ of slack. Before allocating the remaining slack, the M_4 's energy gain is updated to $1700/14uJ/ms$ which is now lower than that of M_3 which is $3107/11 uJ/ms$. As a result, M_3 is now given a slack of $11ms$ to lower its modulation level to $b = 9$. At this point, M_3 and M_4 have the same energy gains, breaking ties arbitrarily, GSS allocates more slack to M_4 leav-

ing 12ms of slack which is unusable by any of the entities in the schedule. The resulting schedule is shown in figure in figure 4. The energy reduction obtained for message M_4 can be calculated as $4807uJ (= 3107 + 1700)$ and that of M_3 is $3107uJ$. Therefore, the GSS algorithms has reduced the energy consumption of the schedule by $7914uJ (= 4807 + 3107)$.

| |
|---|
| <p>Input: Input Schedule, H_i Output: Output Schedule, H_o</p> <ol style="list-style-type: none"> 1 Set $H_o = H_i$; 2 Add all tasks and messages in H_i into the set Q; 3 while $Q \neq \emptyset$ do 4 Pick up the highest energy gain yielding entity e_i from Q; 5 Determine the maximum slack, S_{max} available for e_i; 6 Let s be the time required to operate e_i at its next lower performance level; 7 if $s \leq S_{max}$ then 8 Decrement the performance level of e_i by one level; 9 Update the schedule H_o; 10 end 11 else 12 Remove e_i from Q; 13 end 14 end |
|---|

Algorithm 1: Gain based Static Scheduling Algorithm

3.3 Distributed Slack Propagation Algorithm (DSP)-Dynamic Slack

The actual execution times of tasks exhibit a large degree of variation and are often much lesser than the worst-case estimates used in offline scheduling. As a result, in addition to the static slack that is available offline, some dynamic slack is generated at run-time as the schedule progresses. Such dynamic slack can be utilized to further scale-down the performance levels of the tasks and messages.

Performing global dynamic slack allocation considering all the entities in the schedule requires message passing across nodes via the wireless network which can incur exceedingly high energy and time overheads. With this motivation, we have developed a light weight distributed scheduling algorithm wherein each node independently follows the schedule generated by the GSS algorithm; when the dynamic slack is generated on a processor it is utilized for local tasks and locally originating messages in an independent manner requiring no additional communication across the processors. Such a distributed slack utilization

policy can potentially lead to deadline and precedence constraint violations if the independent decisions taken by each processor are inconsistent. In this section, we first present our DSP scheduling algorithm and then argue its correctness.

Whenever a task T_i completes its execution incurring less time than its worst-case execution time, the generated dynamic slack can be utilized in one of the two ways. The next task, $next(T_i)$ can utilize all of the dynamic slack to reduce its own frequency or it can simply slide left and allow its following task, $next(next(T_i))$ to use the slack. In other words, the task $next(T_i)$ can either use the slack for itself or propagate its slack to the following task $next(next(T_i))$. Similarly, the task $next(next(T_i))$ can choose to propagate the slack to its following task which is $next(next(next(T_i)))$ and so on. In figure 5, task T_1 completes at time 100ms leaving 50ms of slack. Now this slack can be used to either reduce the frequency of T_2 or simply to slide T_2 left so that the frequency of the following tasks' T_6 and/or T_7 can be decreased instead. Choosing the second option has an additional advantage because if T_2 propagates the slack by sliding left, the messages M_1 and M_2 which are produced by T_2 also get the slack (see fig 1.(a)). In the rest of the paper, for each task, we denote the option of using the slack for itself as its *option zero* and the option of propagating the slack further as its *option one*. To decide whether T_2 should choose option zero or one, we need to evaluate the absolute energy reductions (or energy gains) that would be obtained by each of the two options.

In order to capture these different options and estimate their energy gains, we introduce a novel data structure called the *slack propagation tree (SPT)* where each message and task is represented as a separate node in the tree.

3.3.1 Slack Propagation Tree (SPT) Construction

Whenever some dynamic slack is generated due to the early completion of a task T_i on a processor P_j , the processor constructs the SPT with $next(T_i)$ as the root and other local tasks and locally originating messages that are yet to complete as the non-root tree nodes. Consider the SPT shown in figure 6. This SPT is constructed to decide how to use the slack generated by the early completion of task T_1 in figure 5. Since the slack is generated on P_2 , the SPT is constructed with all the tasks on P_2 that are yet to complete (i.e, T_2, T_6, T_7) and their corresponding descendant messages that originate at P_2 (i.e, M_1, M_2).

Two different kinds of edges are used in the SPT. The edges drawn with a label 0 denote the option that the performance of the corresponding entity (from which the edge is drawn) is scaled down and slack cannot be propagated further. Hence the tree terminates here along that branch. On the other hand, edges drawn with a label 1 denote the option that the corresponding entity (from which the edge

is drawn) will propagate the slack further to its following entities. In figure 6, As the slack can potentially be used to reduce the performance level of any of the above tasks and messages after appropriate sliding, a 0-labeled edge is drawn from each one of them to their respective leaf nodes. Whenever slack can be propagated from one task to another and from one message to another, a 1-labeled edge is created as shown in the figure. For example, by sliding T_2 left the slack can be propagated to task T_6 and message M_1 . Hence, 1-labeled edges are from T_2 to T_6 and T_2 to M_1 . In the following, we denote the leaf node of e_k as $\phi_0(e_k)$ and the set of its 1-label child nodes as $\phi_1(e_k)$.

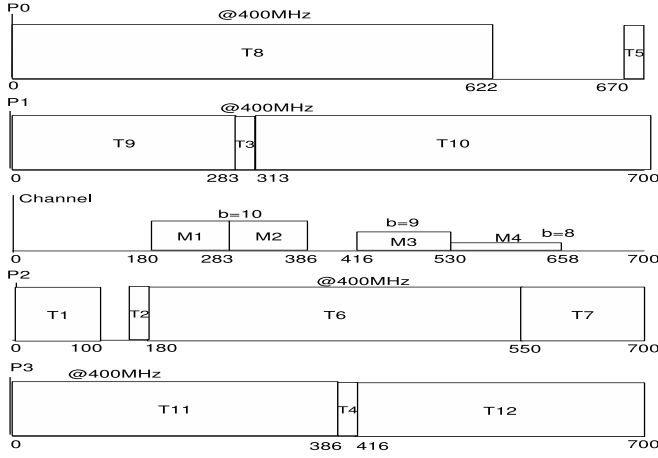


Figure 5. Online Schedule

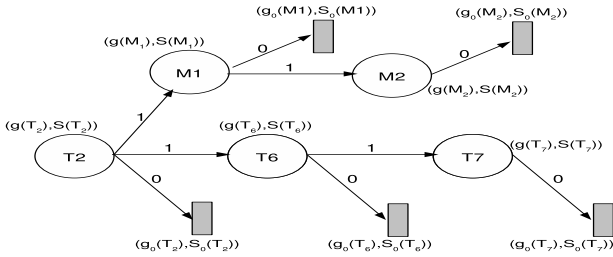


Figure 6. Slack Propagation Tree

The SPT represents different possibilities via which the generated dynamic slack can be utilized. A complete schedule can be obtained by choosing either option zero or option one for each non-leaf internal node in the SPT. In order to make this choice at each node, we need to evaluate the energy gains that would be obtained by each of the two possibilities. To this end, we associate a tuple with each node in the SPT which consists of two elements namely, *energy gain* and *shift value*. Energy gain of a node denotes the amount of energy reduction that would be obtained by choosing the best option (either zero or one) at that node.

The slack required to achieve this amount of energy gain at that node is stored in the shift value of its tuple. We denote the energy gain of a task T_k (message M_j) as $g(T_k)$ (as $g(M_j)$) and the amount of necessary shift as $S(T_k)$ (as $S(M_j)$). For notational convenience, we denote $S(C_0(T_k))$ as $S_0(T_k)$. Similarly, we denote $S(C_0(M_j))$ as $S_0(M_k)$. In figure 6, the tuples for each node are shown by their side. In the following, we present the tuple value calculation details for each node in the SPT.

Tuple value calculations for leaf nodes: The gain value for a leaf node $C_0(e_k)$ denotes the amount of energy reduction that would be obtained by reducing the performance mode of e_k by one level. Mathematically, $g_0(e_k) = E(e_k, p_\alpha) - E(e_k, p_{\alpha-1})$ where p_α denotes the current performance level of e_k . Accordingly, its shift value is calculated as, $S_0(e_k) = T(e_k, p_{\alpha-1}) - T(e_k, p_\alpha)$. If slack is propagated to e_k and it chooses the option zero, its start time is shifted to $st(e_k) - S_0(e_k)$ and its performance is reduced to $p_{\alpha-1}$ in the output schedule while leaving its finish time unchanged. However, if $est(e_k) > st(e_k) - S_0(e_k)$ both $g_0(e_k)$ and $S_0(e_k)$ are set to zero and e_k 's performance is not reduced in the output schedule.

Tuple value calculations for internal nodes: Consider the SPT in figure 6, to decide whether T_2 should choose option zero or one, we compare their corresponding energy gains, $g_0(T_2)$ and $(g(M_1) + g(T_6))$. The option which yields the maximum energy gain is chosen for T_2 . In general the gain value of e_k is calculated as: $g(e_k) = \text{Max}(g_0(e_k), \sum_{e_j \in \phi_1(e_k)} g(e_j))$. If $g_0(e_k) \geq \sum_{e_j \in \phi_1(e_k)} g(e_j)$, $S(e_k)$ is set to $S_0(e_k)$ and e_k chooses option zero. Otherwise, $S(e_k)$ is set to $\text{Max}_{e_j \in \phi_1(e_k)}(S(e_j))$ and e_k chooses option one propagating slack further.

In either case, if the start time of entity e_k cannot be shifted by $S(e_k)$ i.e, if e_k fails the conditions: $est(e_k) \leq st(e_k) - S(e_k)$ its gain and shift values are set to zero and its schedule is left unchanged. If e_k passes this test, its start time and finish time are shifted to $st(e_k) - S(e_k)$ and $ft(e_k) - S(e_k)$, respectively. We note that the above schedule modifications ensure that the new finish times of the entities do not exceed their respective finish times in the input schedule.

3.3.2 The DSP Algorithm

The DSP algorithm uses the SPT data structure to reduce the performance levels of the tasks and messages using the dynamic slack. It proceeds in iterations updating the tuple values of the SPT nodes in each iteration. The following step-by-step procedure outlines the DSP algorithm which is run by each processor independently after recognizing some dynamic slack in the schedule.

| | Iter-1 | Iter-2 | Iter-3 | Iter-4 | Iter-5 |
|---------------|----------|----------|----------|----------|--------|
| $\phi_0(M_1)$ | 3107, 11 | 1700, 14 | 1700, 14 | 0, 0 | 0, 0 |
| $\phi_0(M_2)$ | 3107, 11 | 3107, 11 | 1700, 14 | 1700, 14 | 0, 0 |
| $\phi_0(T_7)$ | 5050, 50 | 0, 0 | 0, 0 | 0, 0 | 0, 0 |
| $\phi_0(T_6)$ | 0, 0 | 0, 0 | 0, 0 | 0, 0 | 0, 0 |
| $\phi_0(T_2)$ | 1830, 10 | 0, 0 | 0, 0 | 0, 0 | 0, 0 |
| T_6 | 5050, 50 | 0, 0 | 0, 0 | 0, 0 | 0, 0 |
| M_1 | 3107, 11 | 3107, 11 | 1700, 14 | 1700, 14 | 0, 0 |
| T_2 | 8157, 50 | 0, 0 | 0, 0 | 0, 0 | 0, 0 |

Table 1. Tuple values after each DSP iteration

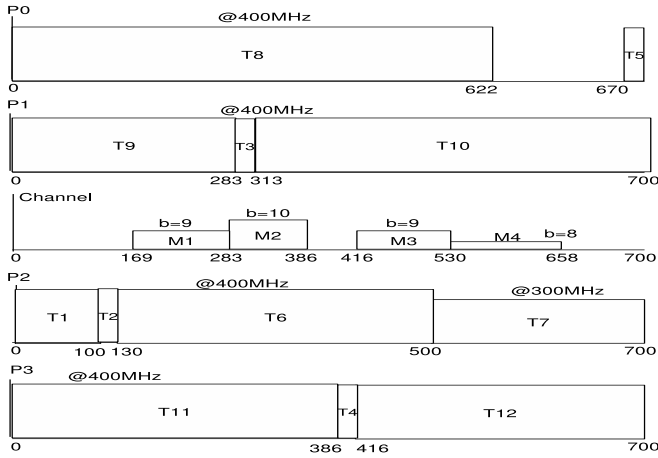


Figure 7. Schedule after one traversal of SPT

In figure 5, when T_2 completes its execution at time $100ms$ generating a dynamic slack of $s = 50ms$, processor P_2 constructs the SPT shown in figure 6 and updates the $ests$ as follows: $est(T_2) = 100$, $est(T_6) = 130$ and $est(T_7) = 500$ while their start times in the current schedule are: $st(T_2) = 150$, $st(T_6) = 180$ and $st(T_7) = 550$. Table 1 shows the tuple values of different SPT nodes across different iterations. The tuple values for the first iteration are shown in column 1. For example, the gain value of $\phi_0(T_7)$ is calculated as $E(T_7, 400MHz) - E(T_7, 300MHz) = 411 * 150 - 283 * 200 = 5050uJ$ and the shift value is calculated as, $T(T_7, 300) - T(T_7, 400) = \frac{400}{300}150 - 150 = 50ms$. The tuple values of the other SPT nodes are calculated in a similar fashion.

The resultant schedule after the first iteration is shown in figure 7. As there is more slack for M_1 and M_2 , at the end of iteration one they show non-zero energy gain values. As a result, DSP proceeds for the next iteration and it terminates after four iterations when the energy gain values of all the nodes reduce to zero. The tuple values for all the iterations are shown in table 1. In the final schedule, M_1 and

Input: Input Schedule, H_i

Output: Output Schedule, H_o

- 1 Construct the SPT including local tasks that are yet to complete and the messages these tasks produce;
- 2 Update the earliest start times of all the tasks and messages in the SPT;
- 3 Calculate the tuple values and determine the best option for each node in the SPT;
- 4 If there is any entity which yields non-zero energy gains proceed to the next step, otherwise exit the DSP algorithm;
- 5 Adjust the schedule according to the options chosen and go to step 3;

Algorithm 2: Gain based Static Scheduling Algorithm

M_2 are assigned a modulation level of $b = 8$ while rest of the schedule remains same as in figure 7. The energy reduction obtained by the DSP algorithm can be easily calculated from the table 1, by adding the unique gain values for each leaf node across several iterations which is $14664uJ$ (obtained as $2 * 3107 + 2 * 1700 + 5050$).

The run-time of the DSP algorithm can be estimated as $O(v^2(k_t + k_m))$ where v denote the number of tasks and messages allocated to the processor of interest. Further, as DSP ensures that each entity in the schedule completes prior to its finish time in the input schedule, each node can safely assume that remote tasks and incoming messages would complete latest by their finish times in the input schedule and thereby, schedule the local tasks and outgoing messages without affecting any precedence constraints. As the ready time and deadline constraints are globally known and fixed parameters they are taken care trivially.

4 Performance evaluation

We performed simulation studies to evaluate the relative performance of the proposed static and dynamic scheduling algorithms. We considered a unit circular region with 15 nodes where node locations are randomly chosen following an uniform distribution. The input schedule was generated using a simplified version of the scheduling algorithm presented in [2] to obtain a feasible schedule for randomly generated input task graphs. For each simulation run, we generated 30 complex periodic tasks and the simulation was performed in the time window of $[0, 50 * LCM]$ where LCM is the least common multiple of the task periods. We varied the following parameters: Channel bandwidth(W), Slack factor (s_f) of a complex task defined as $\frac{deadline-wcet}{T_{trans}}$ where $wcet$ denotes the worst-case execution time of the entire complex task in the input schedule, $\frac{BCET}{WCET}$: ratio

of the best case sub-task execution time ($BCET$) to worst case sub-task execution time ($WCET$). The actual execution time of each sub-task is chosen randomly in the interval $[BCET, WCET]$.

The performance metric in our simulation studies is the normalized total energy consumption of the schedule i.e, the total energy consumption of the schedule normalized with respect to the energy consumption of the input schedule. We compared the following four algorithms in our simulations: (1) Gain based computation only (comp-only) - variation of GSS where only tasks are considered, (2) Gain based communication only (comm-only) - variation of GSS where only messages are considered, (3) GSS, (4) DSP. Where the normalized energy consumption denotes the total energy consumption of the schedule normalized with respect to that of the input schedule.

Effect of channel bandwidth(W): Figure 8 shows the relative performance of the four schemes with varying channel bandwidth. The other parameters are chosen as: $s_f = 10$, $\frac{BCET}{WCET} = 0.5$. At high values of W , messages have significantly high energy gains i.e., reducing the modulation level of a message takes very less additional slack and yields high energy reductions. As a result, *GSS* behaves very much like the comm-only scheme where slack is given only to the messages. At $W = 1000KHz$, GSS shows an improvement of 70% over the comp-only scheme. At low bandwidths like $W = 1KHz$, the message energy gains are comparable to that of the tasks and are sometimes even lower than that of the tasks. Hence the comp-only scheme performs better than the comm-only scheme. Since *GSS* performs a system level slack allocation considering both the tasks and messages it performs better than both the above schemes yielding about 8% improvement over the comm-only scheme. Throughout, DSP performs better than all the above schemes utilizing the dynamic slack in the schedule. At $W = 1000KHz$, it shows an improvement of 15% over the GSS algorithm.

Effect of $\frac{BCET}{WCET}$: Figures 9 and 10 show the relative performance of the above schemes with varying $\frac{BCET}{WCET}$ at $W = 1KHz$ ($s_f = 100$) and $W = 1000KHz$ ($s_f = 0.5$) respectively. In figure 9 with $W = 1KHz$, tasks offer energy gains comparable to that of messages and sometimes offer even better energy gains. As a result, comp-only scheme performs better than the comm-only scheme. The situation is exactly reversed at high bandwidths as shown in figure 10. In all the above scenarios, GSS allocates slack to the highest energy gain yielding entity in the schedule and hence performs better than both the baseline algorithms. In figure 9, it shows an average improvement of 15% over comm-only scheme while in figure 10, it shows an average improvement of 45% over the comp-only scheme. DSP yields further energy savings as shown in figure 10. It shows an improvement of 27% and 19% over the GSS algorithm

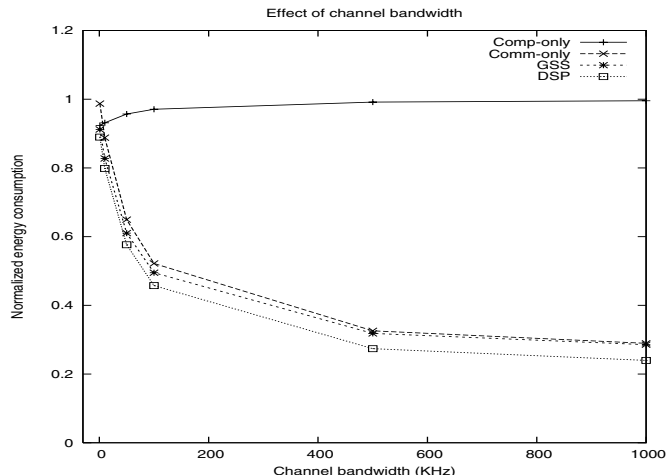


Figure 8. Effect of channel bandwidth

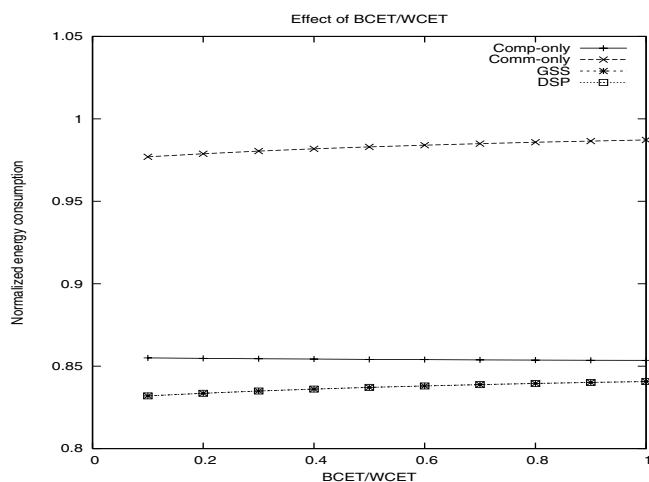


Figure 9. Effect of $\frac{BCET}{WCET}$ at $W = 1KHz$

at $\frac{BCET}{WCET} = 0.2$ and $\frac{BCET}{WCET} = 0.8$, respectively.

Effect of slack factor (s_f): Figure 11 shows the relative performance of the above scheme with varying slack factor with $W = 1000KHz$ and $BCET/WCET = 0.5$. At low s_f , the available slack is insufficient to perform any dynamic voltage scaling and hence comp-only scheme does not show any improvements over the input schedule. On the other hand, as the s_f is increased the available slack is utilized for the tasks and hence comp-only scheme shows an improvement of 6% over the input schedule at $s_f = 200$. Throughout, GSS performs better than both the baseline schemes and shows an improvement of about 50% and 10% over the comp-only and comm-only schemes respectively. DSP uses the ample dynamic slack to reduce the total energy consumption showing an additional improvement of about 15% over GSS at $s_f = 1.0$. As the slack factor increases, most of the tasks and messages operate at low performance levels subsequently, the energy reductions

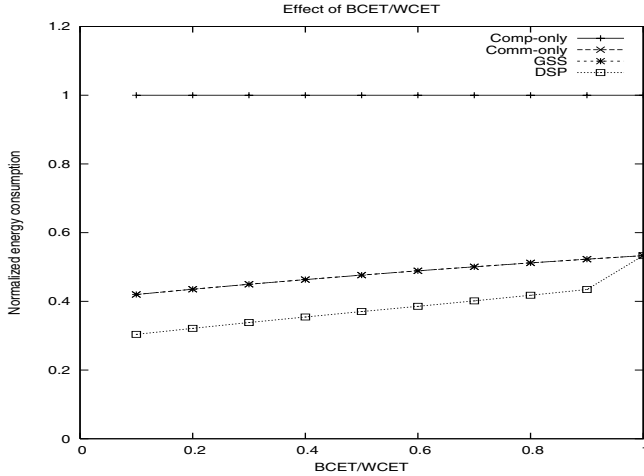


Figure 10. Effect of $\frac{BCET}{WCET}$ at $W = 1000KHz$

brought by the DSP diminish due to the convex nature of the energy function.

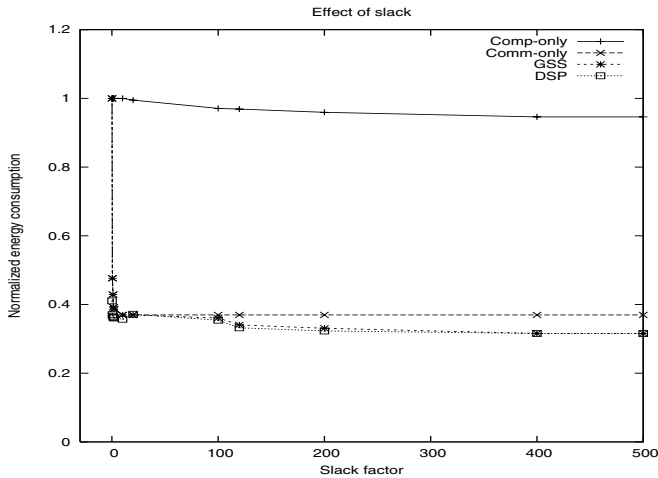


Figure 11. Effect of slack factor

5 Conclusions

Recent technological advancements have opened up a plethora of real-time distributed applications where battery-driven embedded devices with computation and wireless communication capabilities provide time-critical services. Energy management is the key issue in the design and operation of such networked real-time embedded systems. In this paper, we addressed the problem of scheduling complex periodic tasks in a single-hop wireless networked embedded system, where each node supports both DVS and DMS power management techniques. We presented novel centralized static and distributed dynamic scheduling algorithms to solve this problem. Starting with an input feasible

schedule, the proposed algorithms perform slack allocation across tasks and messages with a system-level perspective. We performed simulations to study the effectiveness of the proposed schemes. Our results show the proposed schemes yield significant energy savings over the input schedule.

Several exciting energy-aware research problems can be addressed in this relatively unexplored area of networked real-time embedded systems. Firstly, the problem addressed in this paper can be extended to provide an optimal solution through analytical approaches like integer linear programming formulation. Secondly, a variety of physical layer techniques exist which allow to gracefully tradeoff performance for energy savings. Such techniques include adapting the bit-error-rate and transmission power to the source-destination distance. Each such low-level power management technique defines a new scheduling problem when real-time constraints are considered along with the computation workload in a single-hop network; further, the interplay between the existing processor power management techniques and the communication power management techniques needs extensive investigation. Finally, the above identified energy-aware scheduling problems can be extended to a multi-hop setup to provide end-to-end real-time services.

References

- [1] "Coordination of Safety-Critical Mobile Real-Time Embedded Systems," *Workshop on Research Directions for Security and Networking in Critical Real-Time and Embedded Systems*, San Jose, CA, USA, 2006, apr, TCD-CS-2006-16, <http://www.cs.tcd.ie/publications/tech-reports/reports.06/TCD-CS-2006-16.pdf>
- [2] K. Ramamritham, "Allocation and scheduling of precedence-related periodic tasks," *IEEE Trans. Parallel and Distributed Systems*, vol.6, no.4, pp.412-420, Apr. 1995.
- [3] V. Raghunathan, S. Ganeriwal, C. Schurgers, and M. B. Srivastava, "E2WFQ: An energy efficient fair scheduling policy for wireless systems," in *Proc. of ISLPED*, pp. 30-35, Aug. 2002.
- [4] C. Schurgers, V. Raghunathan, and M. B. Srivastava, "Modulation scaling for real-time energy aware packet scheduling," in *Proc. IEEE Globecom*, pp.3653-3657, Nov. 2001.
- [5] V. Raghunathan, C.L. Pereira, M.B. Srivastava and R.K. Gupta, "Energy-aware wireless systems with adaptive power-fidelity tradeoffs," *IEEE Trans. on VLSI Systems*, Vol.13, no. 2, 2005 pp. 211-225.
- [6] K. Seth, A. Anantaraman, F. Mueller, E. Rotenberg, "FAST: Frequency-Aware Static Timing Analysis," in *Proc. of IEEE RTSS*, pp.40-51, Dec. 2003.
- [7] H. Aydin, R. Melhem, D. Moss & P. Mejia-Alvarez, "Power-Aware Scheduling for Periodic Real-Time Tasks", *IEEE Trans. on Computers*, vol. 53, no. 5, pp. 584 - 600, 2004.
- [8] H. Aydin, V. Devadas and D. Zhu, "System-level Energy Management for Periodic Real-Time Tasks", In *Proc. of IEEE RTSS*, Dec. 2006.
- [9] Bren Mochocki, Dinesh Rajan, Xiaobo Sharon Hu, Christian Poellabauer, Kathleen Otten, and Thidapat Chantem, "Network-Aware Dynamic Voltage and Frequency Scaling", In *Proc. of IEEE RTAS*, April 2007.
- [10] www.intel.com/design/pca/applicationsprocessors/manuals/27878002.pdf