An Online Holistic Scheduling Framework for Energy-Constrained Wireless Real-Time Systems

Thidapat Chantem[†], Jun Yi[†], Shengyan Hong[†], X. Sharon Hu[†], Christian Poellabauer[†], and Liqiang Zhang[‡]

[†]Department of CSE
University of Notre Dame
IN 46556, USA
{tchantem, jyi, shong3, shu, cpoellab}@nd.edu

[‡]Department of CIS
Indiana University South Bend
IN 46634, USA
liqzhang@iusb.edu

***Abstract*—We consider wireless real-time systems that execute computationally-intensive applications and must transmit packets over the network in a timely manner. Existing methods do not consider the importance (i.e., urgency) of a packet as perceived by end users in conjunction with energy consumption, real-time task deadlines, and packet deadlines, inadvertently causing packet priority inversion during transmissions and possibly starvation of some streams. We present an online holistic scheduling framework that explicitly considers packet importance to select packets to transmit and guarantee their deadline requirements using both packet and energy-aware job assignment and scheduling. Our framework is applicable to wireless real-time systems equipped with either a single processor or a multicore system. Based on extensive simulations, we show that our proposed method allows for timely transmissions of the most important packets, which helps to control packet urgency, while saving processor(s) energy.**

I. Introduction

Wireless networks are now common in a variety of applications such as habitat monitoring [14] and surveillance [6]. While many wireless sensor systems typically require minimum hardware to perform lightweight tasks (e.g., periodically waking up to sense and transmit data), powerful processing nodes can be found in certain applications for executing computationally intensive tasks and transmitting packets over the network. Some typical example applications are surveillance and mobile gaming systems. Typically, the performance focus of such systems is the quality of service (QoS) perceived by end users. For example, in a surveillance system, if an intruder enters a premise, it is crucial that not only the equipment captures the intruder on tape, but also that the information is sent to appropriate personnel in a timely manner so necessary actions can be taken. Similarly, gamers in a shooting game would want to see what their opponents are attempting to do from their screen within a certain time window.

For the types of systems described above, due to the time-sensitive nature of these systems, they are usually implemented using real-time tasks, which generate packets to be transmitted over the network. These packets, in turn, often contain time-sensitive materials. To ensure timely packet delivery, high channel utilization, and/or energy efficiency, the system may employ reservation-based channel access protocols, with which they have periodic access to send and receive packets. The time and duration of access may depend on current network conditions (e.g., less often if there are many nodes in the network). Finally, energy consumption of these wireless systems must be minimized since only occasional battery recharge may be possible.

The main problem with the usual implementation of the systems under consideration is that the decision to execute tasks and transmit packets are made independently and changing requirements are not communicated. Specifically, since a node's access to the network may sometimes be limited (due to event storms in sensor networks, high levels of interferences, and varying degrees of network density, for example), fewer packets can be transmitted during that time. While the network scheduler can select the most important packets to send first, the task scheduler, unaware of the situation, may not execute tasks that generate these important packets until later (e.g., when the node no longer has network access).

There is a large body of research on energy minimization for real-time applications, both for uniprocessors (e.g., [18], [24]) and multiprocessors (e.g., [1], [2], [4]). The majority of the work solely focuses on optimizing real-time task performance without any consideration for packet deadlines. At the same time, network-aware work usually focuses on trading network energy with packet latency using packet scheduling and ignores task deadlines, e.g., [12], [21]. The problem of energy-aware scheduling of multiple components has been studied in the past, e.g., [7]. The most relevant papers propose some type of network-aware energy-minimization algorithms for real-time task scheduling [15], [19], which are still task-centric in that packet deadlines are not explicitly considered.

To the best of our knowledge, the work by Yi et al. is the only energy-aware solution that explicitly considers packet deadlines for systems executing real-time tasks [25]. However, this approach treats each packet as equally important, possibly resulting in scenarios where some tasks consistently get their packets transmitted while the packets of other tasks starve. In addition, it is unclear whether or how the work in [25] can be extended to multiprocessor architectures.

One way to ensure that both task and packet deadlines are met is to perform offline analysis to determine how much and when a node requires network access for all packets to be transmitted by their deadlines. However, network conditions unavoidably change and it cannot be guaranteed that each and every node will receive network access as requested. Also, while it is possible to model the relationship between tasks and packets using a directed acyclic graph (DAG), this solution is not viable as solving scheduling problems involving DAGs are usually performed offline due to the high time complexity.

To guarantee the timeliness of important packets that are transmitted over the network, we propose a novel online holistic scheduling framework that considers packet importance, packet deadlines, and job deadlines to judiciously make assignment and scheduling decisions. Specifically, our framework provides a method to select and schedule the more important packets for transmission to address packet urgency while adapting to changing network conditions. The resultant packet schedule is used to derive job deadlines. An LpEDF [24] based scheduling algorithm is used to schedule jobs in uniprocessor systems. For multicore systems, a collection of energy-aware job assignment algorithms is studied. The solutions obtained by each of the components in the framework are evaluated against the optimal solutions and are shown to be a viable alternative to exhaustive search approaches. Comparison with the most closely related work reveals that the proposed framework improves on a specific QoS metric by about 30% on average using about 37% less energy. In addition, our approach reduces window constraint violations [23] by about 57.7%.

This paper is organized as follows. We start by describing the system model and formally define the problem in Section II. Section II-C serves to motivate the need for this work. Section III presents the holistic scheduling framework while the technical details are given in Sections IV and V. Section VI discusses extensions to the framework. Simulation results are given in Section VII and Section VIII concludes the paper.

II. Preliminaries

We now describe our system model and provide some motivations for our work.

A. Task and Packet Model

We consider a set of n independent periodic real-time tasks. Each task τ_i is described by its nominal worst-case execution time C_i, period T_i, and deadline D_i. All tasks are synchronous. The j-th job of task τ_i is denoted by $J_{i,j}$. The absolute release time and deadline of $J_{i,j}$ are $r_{i,j}$ and $d_{i,j}$, respectively. Jobs are executed using EDF [13]. For the rest of the paper, we simply use J_i to denote any individual job of τ_i when it is irrelevant to distinguish between, say, $J_{i,j}$ and $J_{i,k}$.

Without loss of generality, we assume that every job generates a packet at the end of its execution. Packets have firm real-time deadlines, i.e., they must be transmitted by their deadlines or they will be dropped. Packets from different instances of the same task are equal in size while packets from different tasks may vary in size. A packet $S_{i,j}$ is generated by $J_{i,j}$. We simply use S_i to denote a packet generated by any job of τ_i when it is irrelevant to distinguish between, say, $S_{i,j}$ and $S_{i,k}$. A packet S_i is described by its deadline offset X_i, worst-case transmission time Z_i, and importance W_i. Packet importance is a dynamically changing value used to capture the urgency of a given packet. The actual importance of a packet depends on the past transmission history of earlier packets from the same stream. The job J_i, which generates S_i, also inherits that importance level. That is, J_i is more important than J_k if $W_i > W_k$. It is important to note that the importance of a task may be different from one instance to another. Finally, the absolute deadline of S_i is $Y_i = d_i + X_i$ where d_i is the absolute deadline (latest possible finish time) of the job that generates S_i and the deadline offset X_i denotes the relative deadline of packet S_i given d_i. Note that to simplify the analysis, we express Y_i as a function of d_i instead of the release time of S_i since the latter is not a statically known value.

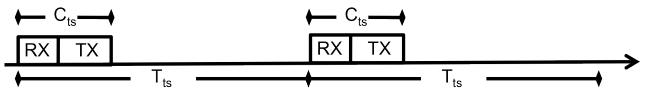


Fig. 1. Network communication model.

B. Hardware and Power Model

For the uniprocessor case, the processor runs at k discrete frequency levels. Note that the maximum frequency level $f_{\max} = f_k$. Each frequency level f_j is described by tuple (P_j, V_j), where P_j and V_j denote the power and voltage when running the processor at frequency level f_j, respectively. For the multicore case, cores are homogeneous and can independently change frequency levels. For now, we assume P_j, $j = 1, \ldots, k$, only consists of dynamic power, which can be expressed as $P_j = \alpha \cdot C_{\mathit{eff}} \cdot V_j^2 \cdot f_j$ where α and C_{eff} are the activity factor and switching capacitance, respectively. In other words, we ignore leakage power for now. This assumption will be relaxed in Section VI-B. The processor energy consumption is defined to be the average power consumption over time. Transition time overhead associated with switching from one frequency level to another have been included in the task worst-case execution times.

The system has a network card. We assume packet transmissions cannot be preempted (so that transmission overheads are reduced). As in [25], the system uses TDMA-like periodic time slots to send and receive packets (Figure 1). Such a network communication model allows for contention-free communication among nodes. No network communication for a particular node takes place outside of its designated TDMA-like time slots. We assume that incoming packets are buffered at the sender and received by the receiver at the beginning of each transmission window (denoted by the RX boxes in Figure 1). The time slots, described by a period T_{ts} and transmission window of length C_{ts}, may change over time to reflect different network usage levels given by the MAC layer protocol. For instance, event storms in sensor networks, high levels of interferences, and varying degrees of network density and traffic due to mobility may cause a node to be temporarily granted less network access (e.g., larger T_{ts} or smaller C_{ts}).

Since our system may be granted less network access due to situations described above, some of its packets may need to be dropped. Note that since packets have firm real-time deadlines, jobs that generate them must generate the packets in time or there is no point in executing them. While it is possible to have scenarios where a job may miss its deadline

TABLE I
EXAMPLE TASK SET

Task	C	T	D	X	Z	W
τ_1	1	2	2	3	1	1
τ_2	1	3	3	3	1	2
τ_3	1	6	6	4	1	3

but its packet is transmitted on time, we do not consider them in this work since we make an implicit assumption that job deadlines are required to ensure data freshness.

The system must minimize its energy consumption to stay alive for as long as possible. In this work, we will focus on the energy used to execute tasks since the energy consumption at the network card is already inherently considered via the periodic time slots (i.e., the network card is turned off or put to sleep outside of C_{ts}) and will not be further discussed.

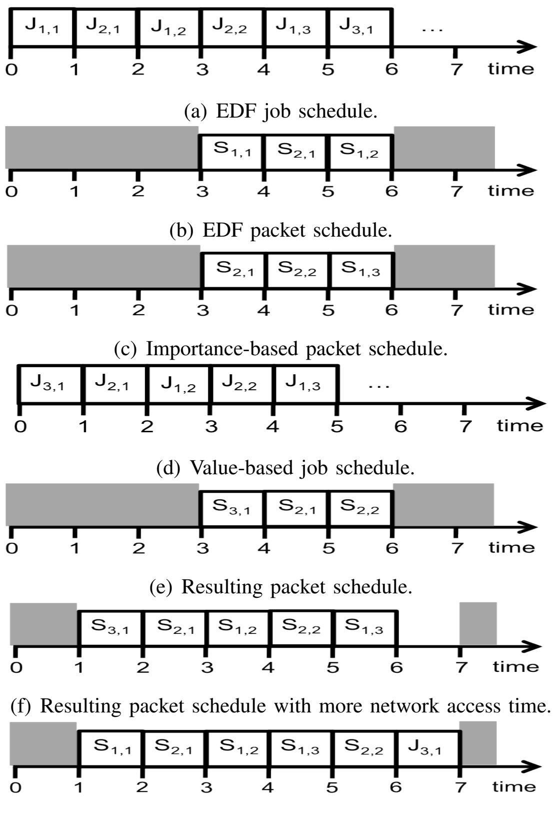
C. Motivation

We use some simple examples to motivate the need for a holistic approach to job scheduling and packet transmission and highlight some key challenges. Assume that we have a set of three tasks (Table I) running on a processor with the maximum normalized frequency level $f_{\max} = 1$. Task (and packet) importance levels are as shown in the last column of Table I, with τ_3 being the most important and τ_1 being the least important. For the sake of clarity, we assume task and packet importance levels are fixed for this example. We further assume here that jobs always require their worst-case execution times. Since the total utilization of the system is 1, the processor executes at the maximum frequency level for the entire duration. The job schedule is shown in Figure 2(a).

Let us assume that the system has network access from time 3 to 6 (and will not get access again until time 12). Using EDF [13], the network schedule is as shown in Figure 2(b). Observe that $S_{3,1}$, despite being the most important packet, is not transmitted.

One solution is to modify the network scheduler to consider packet importance. If the network scheduler were to select the most important packet to schedule first, the resultant packet transmissions are as shown in Figure 2(c). Since the job scheduler is unaware of the needs at the network side, it does not give $J_{3,1}$ a higher priority over the less important jobs. In addition, in both scenarios described so far, many jobs were executed in hope that their packets would be transmitted. This unnecessarily wastes energy.

There exist value-based scheduling algorithms such as [3] in literature. Using a job scheduler that selects the most important job to schedule first, the resultant job and network schedules are as shown in Figures 2(d) and 2(e), respectively. Here, given the network restriction, the value-based job scheduler performs well. However, if the system is granted more access time, say from time 1 to 7, the value-based job scheduler no longer leads to the best packet schedule (Figure 2(f)). In this case, it is possible to transmit all packets in the interval under consideration as shown in Figure 2(g). Note that scheduling algorithms that exploit skip models (e.g., [5] and [23]) suffer from similar shortcomings as value-based scheduling algorithms.



J1,1 J2,1 J1,2 J2,2 J1,3 J3,1 ...
0 1 2 3 4 5 6 7 time
S1,1 S2,1 S1,2
S2,1 S2,2 S1,3
J3,1 J2,1 J1,2 J2,2 J1,3 ...
S3,1 S2,1 S2,2
S3,1 S2,1 S1,2 S2,2 S1,3
S1,1 S2,1 S1,2 S1,3 S2,2 J3,1

(a) EDF job schedule.

(b) EDF packet schedule.

(c) Importance-based packet schedule.

(d) Value-based job schedule.

(e) Resulting packet schedule.

(f) Resulting packet schedule with more network access time.

(g) Alternative packet schedule.

Fig. 2. Job and packet schedules for the example in Table I.

To summarize, without specifically considering packet transmission schedule and network conditions while performing job scheduling, it is not possible to control actual packet transmissions. The situation is even more complicated when packet priority dynamically changes due to past transmission history. A scheduling approach that considers both job scheduling and packet transmission is needed to provide a holistic view of the system and allows for more processor energy to be saved.

III. Holistic Scheduling Framework

To successfully transmit the most important packets by their deadlines while addressing the interdependencies between job scheduling and packet scheduling, we use an adaptive approach. In contrast to traditional job scheduling techniques, we propose selecting and pre-scheduling more important packets to determine the latest packet release times (which is equivalent to determining the deadlines of corresponding jobs) and use these deadlines to make energy-aware job assignment and scheduling decisions.

The flow of our framework is shown in Figure 3 and is especially designed for systems in volatile networks. In the proposed framework, there are four main steps: (i) dynamically assigning packet importance based on past transmission history, (ii) pre-scheduling packets based on current network conditions, (iii) assigning and scheduling jobs, and (iv) actual job execution and packet transmission. These steps are performed periodically (although the actual period can

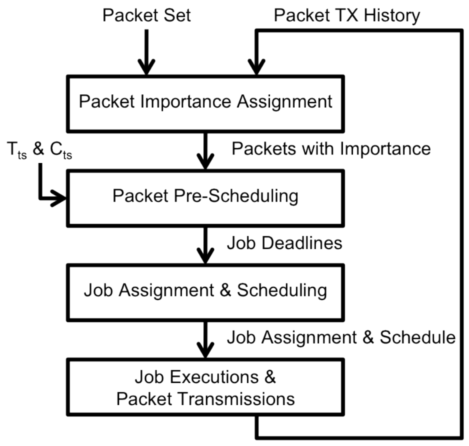

Packet Set
Packet TX History
Packet Importance Assignment
Packets with Importance
T_{ts} & C_{ts}
Packet Pre-Scheduling
Job Deadlines
Job Assignment & Scheduling
Job Assignment & Schedule
Job Executions & Packet Transmissions

Fig. 3. Proposed framework.

change over time to adapt to different network conditions). Specifically, let t be the start time of next network access time interval C_{ts}. In addition, let $t + t_{RX}$ be the time instant when the system may start transmitting packets, where t_{RX} denotes the network access time reserved for receiving packets as shown in Figure 1. We set the current *time interval* $I = [t + t_{RX}, t + C_{ts}]$. The proposed framework will be used prior to the start of every time interval. Note that this selection of the time interval I is reasonable since the value of T_{ts} is usually in the order of several milliseconds, which tends to be much larger than typical real-time task periods (which are in the order of microseconds). That said, if the value of T_{ts} is comparable to task periods, I can be set to encompass several transmission windows of length C_{ts} each.

The primary goal in assigning packet importance is to associate packets with priorities. Assigning importance (i.e., priority) to a packet (or a job) is an old problem that has received significant research attention, e.g., [10], [22], [23]. In general, packets are assigned priorities dynamically based on several considerations such as the usefulness of its content as well as past transmission history. Specifically, whenever a packet misses its deadline, the importance (i.e., urgency) of subsequent packets in the same stream is increased to avoid starvation [23].

The work in [23] provides a way to dynamically and adaptively determine how important a packet is based on past transmission history. Therefore, for the rest of this work, we will assume the method proposed in [23], referred to as DWCS, is used to determine packet importance. Specifically, the current importance level of a task τ_i is defined to be $W_i = \frac{y_i - x_i}{y_i}$, where x_i is the maximum number of packets that can be dropped within a fixed window of y_i packets to ensure that the window constraint is not violated [23]. Both x_i and y_i can change over time [23]. It is crucial to note, however, that our framework can be used in conjunction with any scheme for assigning packet importance. Since we use existing work to assign packet importance, we focus on packet pre-scheduling and job assignment and scheduling in the subsequent sections.

IV. Pre-Scheduling Packets

In this step, the objective is to select and schedule the most important packets for transmission instead of relying on the default packet scheduler used by the network card to make transmission decisions. During the current time interval I, we consider a set of packets Ψ, where $\forall S_i \in \Psi$, the corresponding job release time occurs before the end of I and the absolute deadline of S_i occurs before the start of the next I. Also, for $S_i \in \Psi$, $\forall S_i$, if the original absolute deadline of S_i is greater than $t + C_{ts}$, then it will be set to $t + C_{ts}$.

To determine $\Psi' \subseteq \Psi$ such that Ψ' contains the most important packets and is schedulable, we observe that to minimize the maximum value of dropped packets, the least important packets should be dropped first. The optimal packet set and corresponding schedule can be found by solving a mixed-integer linear program (MILP) with the following objective function.

$$\text{minimize} : \mathit{maxDropped} \tag{1}$$

where

$$\mathit{maxDropped} \geq W_i \cdot (1 - \chi_i), \forall S_i \in \Psi \tag{2}$$

and

$$\chi_i = \begin{cases} 1 & \text{if } S_i \text{ is not dropped} \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Let Ω' be the job set containing all jobs that generate the selected packets in Ψ'. Given a packet schedule, the transmission start time TXS_i of a packet S_i, $\forall S_i \in \Psi'$ is known. This transmission start time in fact coincides with the latest time that job J_i must finish executing. In other words, for packet S_i, $\forall S_i \in \Psi'$, to be transmitted on time, the new absolute deadline d'_i of J_i is $d'_i = \min\{d_i, TXS_i\}$. We also define the transmission finish time $TXF_i = TXS_i + Z_i$. In the MILP formulation, the variables we would like to solve for is the modified absolute job deadlines d'_j, $\forall J_j \in \Omega'$. We also define the following variables.

$$\eta_{i,j} = \begin{cases} 1 & \text{if } d'_j \geq d'_i \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

$$\sigma_{i,j} = \begin{cases} 1 & \text{if } TXS_i \leq TXS_j \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

The value of the actual deadline of each job must satisfy the following two constraints, one of which represents the necessary conditions for job schedulability.

$$d'_j \leq d_j, \forall J_j \in \Omega, \tag{6}$$

$$r_j + C_j \leq d'_j, \forall J_j \in \Omega. \tag{7}$$

Each packet must meet its deadline and cannot start until its release time at the earliest. That is,

$$\forall S_j \in \Psi$$

$$TXF_j = TXS_j + Z_j \cdot \chi_j, \tag{8}$$

$$Y_j \geq TXF_j, \tag{9}$$

$$d'_j \leq TXS_j + (1 - \chi_j) \cdot \Lambda, \tag{10}$$

where Λ is some large constant.

The following constraints are used to ensure that packet transmissions do not overlap.

$$\begin{aligned}
&\forall S_i, S_j \in \Psi, S_i \neq S_j \\
&\quad TXF_i \leq TXS_j + (1-\sigma_{i,j}) \cdot \Lambda \\
&\qquad + (1-\chi_j) \cdot \Lambda + (1-\chi_i) \cdot \Lambda, && (11)\\
&\quad TXF_j \leq TXS_i + \sigma_{i,j} \cdot \Lambda + (1-\chi_j) \cdot \Lambda \\
&\qquad + (1-\chi_i) \cdot \Lambda. && (12)
\end{aligned}$$

The following constraints are needed to ensure that the variables $\eta_{i,j}$, and $\sigma_{i,j}$, $\forall S_i, S_j \in \Psi$, are as defined.

$$\begin{aligned}
&\forall S_i, S_j \in \Psi \\
&\quad 1 \leq \eta_{i,j} + \eta_{j,i}, && (13)\\
&\quad d'_i \leq d'_j + (1-\eta_{i,j}) \cdot \Lambda, && (14)\\
&\quad d'_j \leq d'_i + \eta_{i,j} \cdot \Lambda - \epsilon, && (15)\\
&\quad \epsilon \leq d'_j - d'_i + \eta_{j,i} \cdot \Lambda, && (16)\\
&\quad 1 \leq \sigma_{i,j} + \sigma_{j,i}, && (17)\\
&\quad TXS_i \leq TXS_j + (1-\sigma_{i,j}) \cdot \Lambda, && (18)\\
&\quad TXS_j \leq TXS_i + \sigma_{i,j} \cdot \Lambda, && (19)
\end{aligned}$$

where ϵ is some small constant used to prevent numerical errors due to floating point computations.

Some readers may have noticed that when the packet set is selected and packet schedule is determined, the job schedule is unknown. Since jobs generate packets, it is difficult to guarantee that the resultant packet schedule is indeed achievable (i.e., whether the job set Ω' with modified deadlines is in fact schedulable). We observe that from the job scheduling perspective, a job set is more likely to be feasible if its jobs have more laxity (i.e., more time to finish from the time they are released). For this reason, it may be desirable to solve an additional MILP formulation once the select packet set Ψ' is obtained with the following objective function.

$$\text{maximize} : minLax \qquad (20)$$

where

$$minLax \leq d'_j - r_j - C_j, \forall J_j \in \Omega'. \qquad (21)$$

Alternatively, some type of weighted sum trading off dropped packets against job laxity can be used in order to solve only one MILP formulation. In this work, we choose to solve two MILP formulations since the weight selection can be arbitrary.

While the MILP solver is guaranteed to return an optimal solution, it is not suitable for online use when the problem size is large since it is too computationally intensive. We therefore introduce our simple heuristic. To maximize job laxities, packets should be scheduled as late as possible. We propose scheduling packets with later deadlines first to maximize laxity. If two or more packets share the same deadline, ties are broken in favor of the packet S_i with the largest $r_i + C_i$, again, to maximize laxity. We keep track of the end of the current schedule using the variable t_{end}. That is, a packet cannot be scheduled after t_{end}. However, if a packet deadline occurs before t_{end}, then its transmission finish time will be set to its deadline. Algorithm 1 shows the steps required to select Ψ' and derive a packet schedule that is guaranteed to be feasible and that attempts to maximize job laxities.

Algorithm 1 Packet_Schedule(Ψ)

```
1: sort $\Psi$ in non-increasing order of deadlines, ties broken
   in favor of the packet $S_i$ with the largest $r_i + C_i$
2: $droppedSet \leftarrow \emptyset$
3: $done \leftarrow false$
4: while $done = false$ do
5:   $t_{end} \leftarrow Y_0$ // $Y_0$ is the absolute deadline of the first
     packet in $\Psi$
6:   for each $S_i \in \Psi$ do
7:     if $t_{end} \geq Y_i$ then
8:       $TXS_i \leftarrow Y_i - Z_i$ // schedule $S_i$ by $Y_i$
9:       $TXF_i \leftarrow Y_i$
10:    else // Other packets have been scheduled later on so
       schedule $S_i$ now
11:      $TXS_i \leftarrow t_{end} - Z_i$
12:      $TXF_i \leftarrow t_{end}$
13:    $t_{end} \leftarrow TXS_i$
14:  $flag \leftarrow false$
15:  for each $S_i \in \Psi$ do
16:    if $r_i + C_i > TXS_i$ then // $r_i + C_i$ is the earliest time
       $S_i$ can be generated
17:      $flag \leftarrow true$
18:  if $flag = true$ then
19:    $\Psi \leftarrow \Psi - S_q$ // $S_q$ is the packet of lowest importance
20:    $droppedSet \leftarrow droppedSet \cup S_q$
21:  else
22:    $done \leftarrow true$
23: $\Upsilon \leftarrow$ packet schedule represented by a linked list sorted
   in non-increasing order of transmission start times // The
   node for packet $S_i$ is tagged with $TXS_i$ and $TXF_i$
24: sort $droppedSet$ in a non-increasing order of importance
25: for $S_i \in droppedSet$ do
26:   for each node $N_j \in \Upsilon$ do
27:     $N_k \leftarrow N_j.nextNode$
28:     if $N_k \neq \emptyset$ then
29:       $s \leftarrow N_k.TXF$
30:     else
31:       $s \leftarrow t + t_{RX}$
32:     $f \leftarrow N_j.TXS$
33:     if $f \leq Y_i$ and $f - s \geq Z_i$ and $s \geq r_i + C_i$ then
34:       $TXS_i \leftarrow s$
35:       $TXF_i \leftarrow f$
36:       $\Psi \leftarrow \Psi \cup S_i$
37:       break
```

When two or more packets share the same deadline, the packet S_i, which has a larger $r_i + C_i$, will start later than the packet S_j with a smaller $r_j + C_j$ to maximize laxity. Note that Algorithm 1 ensures that the constraints in (6) and (7) are satisfied to ensure that the resultant packet transmission start times, and hence the deadlines of the corresponding jobs, are valid. Specifically, the constraint in (6) is ensured by Lines

7–9 of Algorithm 1. On the other hand, the constraint in (7) is ensured by Line 16. That is, if there exists a packet S_i which is set to be transmitted before the job that generates it can possibly finish executing, the least important packet is dropped and the schedule is reconstructed.

One problem that may arise with the above method of finding a packet schedule is that packets with lower importance may be dropped, even when it is not necessary. Specifically, consider two packets S_i and S_j where $W_i < W_j$. If S_j is dropped, then S_i is also dropped. While this dropping technique does not affect the maximum value of the most important packet dropped (see (2)), it is still desirable to execute S_i in the current time interval (if possible) since doing so may prevent an increase in W_i in the next time interval. For this reason, after a packet schedule has been obtained, we test to see if some of the dropped packets can be restored, as shown in Lines 23–37 of Algorithm 1. The worst-case running time of Algorithm 1 is in $\mathcal{O}(|\Psi|^2)$.

V. Energy-Aware Job Assignment and Scheduling

The third component of our framework focuses on energy-aware job assignment and scheduling to meet as many job deadlines as possible for minimizing the maximum value of the most important packet that is dropped (equation (2)) while saving energy. We start by considering uniprocessor architectures before moving on to multicore systems.

A. Uniprocessors

In a single processor system, there is no need to consider how to assign jobs, only how to schedule them. Recall that from the last section, we are provided with a job set Ω', which generate the packets to be transmitted. Our goal here is to schedule as many jobs in Ω' as possible by their deadlines while minimizing energy.

While our tasks are originally periodic (Section II), job deadlines may be shortened in the packet pre-scheduling phase. In other words, for the job set under consideration, each job has a release time, worst-case execution time, and deadline. To schedule these jobs, we propose using LpEDF [24], which has been proved to be optimal in terms of minimizing energy consumption of ideal processors. (For discrete-speed processors, two speed levels can be used to approximate the desired speed level, if the latter is not available [11]). LpEDF determines a frequency schedule for a given job set. Observe, however, that the job set Ω' is not necessarily schedulable, i.e., the resultant frequency schedule after applying LpEDF may contain frequency levels that are higher than $f_{\max}$. In such a case, some jobs will need to be dropped.

Since jobs inherit the importance of the packets they generate, determining which jobs (and therefore packets) to drop in this step can be accomplished in the same manner as selecting the most important packets in Algorithm 1. Therefore, we omit the repeated explanation but provide our job scheduling algorithm in Algorithm 2.

We now discuss the performance of Algorithm 2 in the following theorem. The proof is trivial and is thus omitted.

Algorithm 2 Uniprocessor_Job_Sched(Ω')

```
1: $\Pi \leftarrow$ job schedule obtained from $\Omega'$ using LpEDF [24] // each entry in $\Pi$ is tuple ($e_i$, $J_i$) where $e_i$ and $J_i$ denote the frequency level to be used for executing job $J_i$
2: while $\exists e_j \in \Pi : e_j > f_{\max}$ do
3:     $\Omega' \leftarrow \Omega' - J_q$ // $J_q$ is the job whose packet is of lowest importance
4:     $\Pi \leftarrow$ job schedule obtained from $\Omega'$ using LpEDF [24]
5: return $\Pi$
```

Theorem 1: For a given job set Ω' running on an ideal processor, Algorithm 2 minimizes the value of the most important packet that is dropped.

The worst-case time complexity of Algorithm 2 is $\mathcal{O}(|\Omega'|^3 \log^2|\Omega'|)$ since LpEDF requires $\mathcal{O}(|\Omega'|^2 \log^2|\Omega'|)$ to run in the worst-case [24] and there can be at most $|\Omega'|$ iterations of the while loop. The time complexity of Algorithm 2 can be reduced to $\mathcal{O}(|\Omega'|^2 \log^3|\Omega'|)$ if binary search is used in the while loop instead of linear search. As with Algorithm 1, to prevent jobs with lower importance from being dropped even when it is unnecessary, a test can be performed to see if some of the dropped jobs can be restored. This addition is very similar to Lines 23–37 of Algorithm 1 and increases the worst-case running time of Algorithm 2 to $\mathcal{O}(|\Omega'|^3 \log^3|\Omega'|)$.

B. Multicore Systems

In a multicore system, we first need to assign $J_i \in \Omega'$, $\forall J_i$, to cores before scheduling can take place. The problem of task assignment is NP-hard in general, except for frame-based tasks where all tasks share a common deadline [4]. Most existing work assume periodic tasks and fixed deadlines (e.g., [16]). Recall that the jobs under consideration have release times, worst-case execution times, and deadlines. In addition, job deadlines may be smaller than their implicit deadlines and the deadlines are not necessarily constant from one task instance to another. There exist research results on job-level assignment, e.g., [20], but they do not consider energy.

We focus on the energy-aware job assignment phase since once it is complete, the scheduling technique discussed in the last section can be straightforwardly applied on each processor. The state-of-the-art approach to assigning jobs to cores is to perform load balancing, as in [4]. To reduce energy, jobs should be assigned to cores in such a way so as to balance the energy consumption among cores. In the absence of significant leakage power, it is well known that less energy is consumed if the processor runs as slow as possible (i.e., using the lowest possible frequency levels). We propose to study a collection of heuristics (three of which are our own) whose performance will be assessed in Section VII-B. In Section VI-B, we discuss how our approach can be extended to processors in which leakage power is significant.

- **Largest-Job First (LJF) [4]**: Sort jobs in a non-decreasing order of worst-case computation time and assign a given job to the core with the least aggregated computation time.

- **Largest-Density First (LDF) [4]**: Same as LJF except that job densities are used instead of job worst-case computation times. The density of a job J_i is defined as $\delta = \frac{C_i}{d_i - r_i}$.
- **Most-Important First v.1 (MIF-1)**: Sort jobs in a non-decreasing order of importance and assign a given job to the core with the least intensity during the job's active interval, which starts at the job's release time and ends at the job deadline. The intensity of the active interval of length a of job J_i is $\frac{\sum_{J_k \in \Pi} C_k \cdot ov_{k,i}}{a}$, where Π is the set of jobs currently scheduled on the core under consideration and $ov_{k,i}$ is the ratio of the overlap in active intervals between J_k and J_i.
- **Most-Important First v.2 (MIF-2)**: Same as MIF-1 except that a given job is sent to the core with the least aggregated densities (similar to LDF).
- **Most-Important First v.3 (MIF-3)**: Same as MIF-1 except that a given job is sent to the core with the least aggregated importance.

Except for MIF-1, which runs in $\mathcal{O}(|\Omega'|^2 \cdot M)$, where M is the number of cores in the system, all other algorithms presented above have a worst-case time complexity of $\mathcal{O}(|\Omega'| \cdot \log |\Omega'| + |\Omega'| \cdot M)$. Once all the jobs in Ω' have been assigned, Algorithm 2 can be used on each core to determine job schedules (and possibly drop some jobs).

VI. Notes on Framework

We now discuss some generalizations and limitations of the proposed framework.

A. Extensions to Task and Packet Models

In Section II, we made some simplifying assumptions with regards to the system model to simplify the discussion of the proposed framework. Specifically, we assumed that all jobs generate packets and packets are generated at the end of job execution. We first discuss the inclusion of non-packet generating tasks in our framework. If there are non-packet generating, soft real-time tasks in the system, they can be assigned and scheduled when the system is not busy executing packet generating jobs. On the other hand, the inclusion of hard real-time non-packet generating tasks in the system is part of our ongoing investigation.

The assumption that each packet is generated at the end of job execution is in fact not required in our framework. All analyses from the previous sections hold with regards to packet feasibility and job schedules; if a job generates a packet before it finishes executing, that packet arrives to the network queue earlier and has no adverse effect on the packet schedule (although it may increase the network buffer size). That said, more energy can be saved if it is known exactly when a job will generate a packet since job deadlines set in Section IV can be extended accordingly.

Similarly, though all the algorithms implicitly assume that jobs (packets) require their worst-case execution times (transmission times), all the analyses in this paper remain valid. Obviously, more energy can be saved if slack reclamation is used but this topic is beyond the scope of this work.

B. Leakage Considerations

Due to shrinking device sizes and aggressive voltage scaling, subthreshold leakage current increases exponentially as the supply voltage is reduced, causing energy to be wasted when the system runs at very low frequency levels [9]. For such systems, running the processor below the critical frequency $f_{critical}$ is suboptimal [9] and can cause the system to consume more energy than necessary. While it is possible to adjust an existing frequency schedule by substituting any frequency level $f_j < f_{critical}$ with $f_{critical}$, a more energy-efficient schedule may exist.

There is a wealth of research on leakage-aware energy minimization for real-time systems, e.g., [9], [17]. In particular, the work by Niu and Quan [17] can be used on each processor after Algorithm 2 to obtain a leakage-aware schedule.

VII. Evaluation

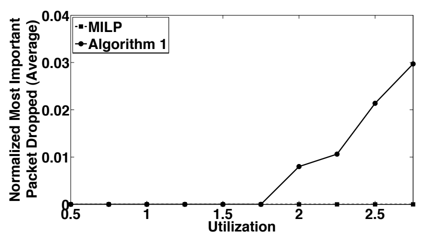
This section presents simulation results to demonstrate the effectiveness of our framework. We evaluate the performance of the packet pre-scheduling and job assignment and scheduling steps, and compare our framework with an existing work that solves a similar problem.

A. Pre-Scheduling Packets

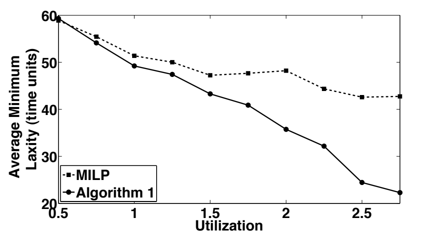
Recall that the primary goals of Algorithm 1 are to select the most important packets to schedule and derive a schedule that maximizes the minimum job laxity. We compared the performance of our heuristic (Algorithm 1) with that of the MILP, which was solved using CPLEX with AMPL. The time limit was set to 1 minute for the CPLEX solver, which means that if an optimal solution has not been found within that time, the best feasible solution found so far will be returned.

For our benchmarks, 100 tasks sets consisting of between 10-45 tasks each were randomly generated for 10 different utilizations ($U = 0.5, 0.75, \ldots, 2.75$) with a total of 1000 task sets overall. The utilization is defined to be $\sum_{i=1}^{n'} \frac{C_i}{T_i}$ where n' is the total number of tasks in the system. Task periods ranged from 50 to 300 time units, with the worst-case execution time being set between 30-60% of the period. The absolute deadline of a packet is set to be equal to the absolute deadline of the corresponding job. Packet sizes vary randomly between 1-3.5% of C_{ts}, which is set to 300 time units. Packet importance is between $[0, 1]$. Finally, $|I| = 300$.

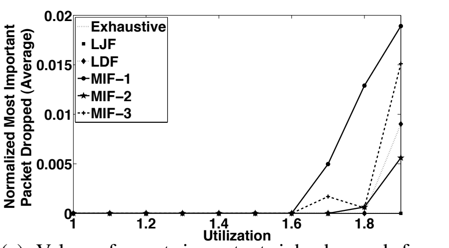
The first set of results are summarized in Figure 4(a), which shows the value of the most important packet dropped (i.e., the value of equation (2)) on the y-axis as a function of system utilization. When the utilization is low, there are fewer tasks in the system and hence fewer packets. On the other hand, more packets are generated when the utilization is high because there are more tasks in the system. The MILP was able to keep every packet while our heuristic approach dropped some of them. That said, Algorithm 1 dropped very few packets and only the least important ones. On average, the normalized value of the most important packet dropped is 0.0073 (0.73% deviation on average in other words).



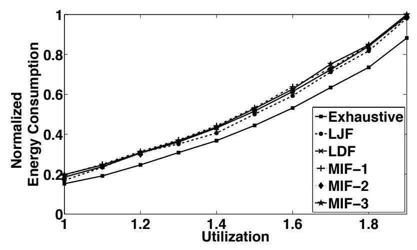
(a) Value of most important packet dropped from different packet scheduling algorithms.



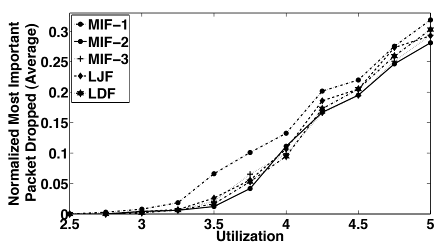
(b) Minimum job laxity of different packet scheduling algorithms.



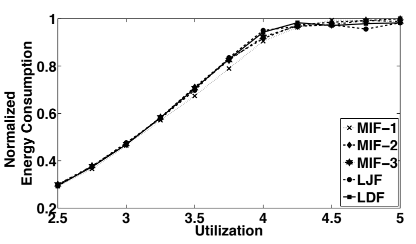
(c) Value of most important job dropped from different job assignment algorithms for the two-core case.



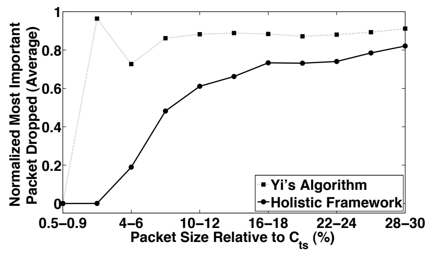
(d) Energy consumption of different job assignment algorithms for the two-core case.



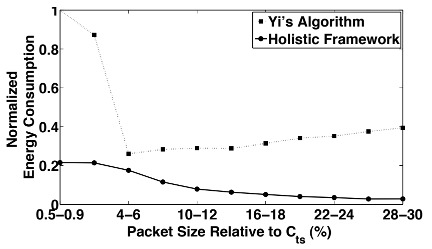
(e) Value of most important job dropped from different job assignment algorithms for the four-core case.



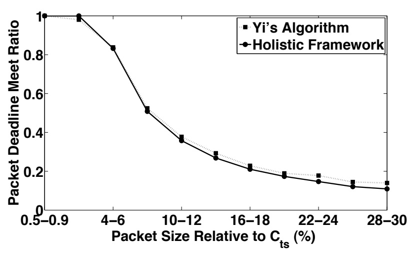
(f) Energy consumption of different job assignment algorithms for the four-core case.



(g) Comparison to [25] in terms of minimizing the maximum importance of dropped packets.



(h) Comparison to [25] in terms of energy consumption.



(i) Comparison to [25] in terms of packet deadline meet ratio.

Fig. 4. Simulation results.

In terms of maximizing the minimum job laxity (Figure 4(b)), Algorithm 1 performs very well when the total task utilization is low (and fewer packets are generated). As the total task utilization increases, Algorithm 1 deviates more from the results obtained by solving the MILP formulation. Specifically, the average minimum laxity from Algorithm 1 is about 17.68% smaller than that of the MILP and up to 47.59%. It must be emphasized, however, that Algorithm 1 performs very well in terms of minimizing the value of the most important packet dropped. In addition, while the MILP solver can be used for very small problem instances, it is too computationally intensive for solving medium to large problem instances online (and it cannot guarantee that any feasible solution will be found within some time limit). On average, Algorithm 1 is at least 1,000× faster than the MILP.

B. Job Assignment & Scheduling

In this section, we assess the performance of the proposed energy-aware job assignment and scheduling algorithms presented in Section V-B. We compared our solutions with the optimal solutions obtained by a brute-force algorithm. In an exhaustive search, all possible job assignments are explored and the job assignment that is both feasible and results in the least amount of energy consumed is identified.

In the simulations, we assume a multicore system consisting of two identical cores. Each core is modeled after the Intel Core 2 Duo [8], with a maximum power consumption of 65 W and seven normalized frequency levels: 0.462, 0.615, 0.692, 0.769, 0.846, 0.923, and 1. System utilization ranges from $1.0, 1.1, \ldots, 1.9$. A total of 100 task sets were generated for each utilization (2000 task sets in total). The same method and parameters as in the last section were used to generate the task sets. Since the total number of job assignment combinations is $|M|^{|\Omega|}$ where $|M|$ is the number of cores and $|\Omega|$ is the number of jobs, we limit the number of jobs to 20 to stay within the range of a long integer (this is also the reason why we limit the number of cores to only two). Specifically, we discarded any task sets containing more than 20 jobs during I ($|I| = 300$) and regenerate them until 100 task sets were found for each utilization.

Figure 4(c) shows the maximum importance level of dropped jobs (and hence packets) as a function of system utilization using the different job assignment algorithms. With only two cores and at most 20 jobs, LJF yields the best results (same as the exhaustive search), with no dropped jobs, while MIF-1 performs the worst. In terms of energy consumption (Figure 4(d)), no heuristic obtains the minimum energy consumption that the exhaustive search achieves, though LJF again yields the best results among the different heuristics, followed closely by MIF-2 and LDF.

To assess the performance of the different job assignment algorithms when the problem size is larger (and where an exhaustive search cannot find solutions within an hour), we performed an additional set of simulations. The system is assumed to have four identical cores, which are again modeled after the Intel Core 2 Duo [8]. The system utilization ranges from $2.5, 2.75, \ldots, 5$. A total of 100 task sets were generated for each of the utilization (1100 task sets in total).

Figure 4(e) shows the maximum importance level of dropped jobs as a function of system utilization using different job assignment algorithms discussed in Section V-B. Overall, MIF-2 and MIF-3 yield the best results, though they can be outperformed by LJF and LDF in some cases. MIF-2 and MIF-3 improve on the value of most important job dropped by about 1.08-2.67% on average when compared to LJF and LDF. In terms of energy consumption, MIF-2 and MIF-3 once again outperform LJF and LDF on average (though not by much, as shown in Figure 4(f)).

Based on the simulation data, MIF-2 provides the best performance for a given energy consumption level, although the performance of all five algorithms are close, which may suggest the limits of this type of job assignment algorithms. In any case, since the brute-force approach takes several minutes to find the optimal solution for even a small benchmark, it cannot be used online. Based on the results given above, MIF-2 performs well enough (and is 100,000× faster than the exhaustive search on average) to be a viable alternative.

C. Entire Framework

We compare the effectiveness of our proposed framework with the most closely related work by Yi et al. [25]. In Yi's algorithm, both packet and job deadlines are considered but all packets are assumed to be equally important. Simulations were performed assuming a single processor core is used since the work in [25] only focuses on uniprocessor scenarios. The task sets and other parameters (including packet importance) are generated in the same way as in Sections VII-A and VII-B except for the following parameters. The average system utilization is set to 0.5 and the size of a single packet varies randomly between 0.5-30% of C_{ts}. The larger the packet size, the more loaded the network is and the harder it is to transmit all packets by their deadlines.

The graph in Figure 4(g) shows the maximum importance level of dropped packets as a function of packet size. As expected, our proposed holistic framework always outperforms Yi's algorithm [25], which does not consider packet nor job importance. On average, our framework reduces the average maximum importance of dropped packets by 30% (up to 96.4%). As the network becomes more overloaded, our framework is forced to drop more packets (but always the least important ones first). This is the reason why the curve representing the holistic framework tends to increase as the packet size increases. As for Yi's algorithm, it is hard to predict which packets will be dropped since all packets are considered to be equally important. For instance, Yi's algorithm often results in important packets being dropped (Figure 4(g)). However, at times, it may by chance transmit more important packets instead of less important ones (i.e., the dip in Figure 4(g)).

TABLE II
ADDITIONAL PARAMETERS FOR SIMULATIONS OVER SEVERAL TIME INTERVALS

Time Interval	C_{ts}	T_{ts}	Packet size increase
$[0 : 5,000]$	300	500	0%
$[5,000 : 10,000]$	100	500	0%
$[10,000 : 15,000]$	300	500	10%

As expected, our proposed framework also results in less energy consumed (Figure 4(h)). When the network is underloaded (shown towards the left side of Figure 4(h)), our method pre-schedules jobs and packets ahead of time to maximize job laxity and hence reduces energy consumption. In contrast, Yi's algorithm greedily tries to send out packets as soon as possible, even when it is unnecessary. This results in large energy consumption. As the network becomes more loaded, our approach saves energy by dropping jobs whose packets are never transmitted. On the other hand, Yi's algorithm always executes jobs, even if the corresponding packets are dropped. The dip in the curve representing Yi's algorithm can be explained as follows. As the network becomes more loaded, the next available transmission time becomes further away in the future, allowing jobs to be executed at lower speed. As the packet size increases, however, more energy will be consumed because jobs need to finish earlier to send larger packets out. This is the reason why the curve slowly increases again after the dip. In summary, our approach saves energy by 37.3% on average when compared to Yi's algorithm (up to 78.5% and at least 10%).

An often used performance metric for comparing packet scheduling algorithms is the deadline meet ratio, which is the ratio between the number of packets transmitted by the deadline and the total number of packets during some time interval. As shown in Figure 4(i), on average, Yi's algorithm sends out 1.7% more packets on average and up to 3.1% when compared to our framework. However, as will be shown at the end of this section, our approach results in a higher deadline meet ratio when the behavior of the system over a long period of time is considered.

Finally, to show that our approach is able to control packet urgency over time, we performed a set of simulations where task importance is dynamically adjusted in each window using the rules in [23]. A total of 100 task sets consisting of 10 tasks each were simulated for 15,000 time units. The task set utilization is 0.5 and the average packet sizes vary randomly between 0.5-2% of the C_{ts}. For the DWCS window constraint [23], each task starts with a y value of 20. The starting x value varies randomly between 1 and 19. The DWCS window constraint is set to $\frac{x}{y}$, which indicates that at most x packets can be dropped within a window of y packets. A violation occurs when more than x packets are transmitted in a window of y packets for any given stream. The values

TABLE III
RESULTS FOR SIMULATIONS OVER SEVERAL TIME INTERVALS

	Yi's Algorithm [25]			Holistic Framework		
	avg	max	min	avg	max	min
No. of violations	350.2	581.0	196.0	148.3	402.0	1.0
Norm. energy (%)	84.7	100	73.0	30.3	40.8	24.5
Deadline meet ratio	0.47	0.56	0.38	0.71	0.88	0.59

of x and y are updated every time interval I. In general, the value of y for a stream is decreased if its previous packet is serviced before its deadline. For more information, readers are referred to Figures 2 and 3 in [23]. Additional parameters used in this set of simulations are shown in Table II. The last column of Table II denotes the increase in packet sizes to simulate changing transmission rate.

Since we are using the window constraint in [23] as the performance metric, the goal is to minimize the number of window constraint violations while saving energy. As shown in Table III, our approach significantly improves upon Yi's algorithm both in terms of the number of window constraint violations and energy by intelligently dropping some packets. Also, since the holistic framework only executes jobs that are necessary, it offers significant energy savings compared to Yi's algorithm. Finally, Table III shows that our method of pre-scheduling packets in each individual time interval helps to improve the total deadline meet ratio since it does not suffer as much from the "domino" effect when the network is overloaded. This phenomenon is not reflected in Figure 4(i) since the latter only counts packets whose deadlines fall within that time interval and cannot capture the full benefit in using our adaptive approach.

VIII. Summary

To increase the performance of computationally-intensive wireless real-time applications in volatile networks, we presented a holistic scheduling framework that considers packet and job deadlines, as well as packet importance, to ensure timely transmissions of the most important packets while saving processor(s) energy. The proposed framework can adapt application performance to changing network conditions and is shown to outperform the best existing technique by about 30% on average. Our future goals are to implement the proposed framework on a real system and extend it to consider processor transition overheads and heterogeneous multicore systems.

References

[1] J. Anderson and S. Baruah, "Energy-efficient synthesis of periodic task systems upon identical multiprocessor platforms," in *Proceedings of the International Conference on Distributed Computing Systems*, Mar. 2004, pp. 428–435.

[2] H. Aydin and Q. Yang, "Energy-aware partitioning for multiprocessor real-time systems," in *Proceedings of the International Symposium on Parallel and Distributed Processing*, Apr. 2003, p. 113.2.

[3] G. Buttazzo, M. Spuri, and F. Sensini, "Value vs. deadline scheduling in overload conditions," in *Proceedings of the Real-Time Systems Symposium*, Dec. 1995, pp. 90–99.

[4] J.-J. Chen, C.-Y. Yang, H.-I. Lu, and T.-W. Kuo, "Approximation algorithms for multiprocessor energy-efficient scheduling of periodic real-time tasks with uncertain task execution time," in *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, Apr. 2008, pp. 13–23.

[5] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m,k)-firm deadlines," *IEEE Transactions on Computers*, vol. 44, no. 12, pp. 1443–1451, Dec. 1995.

[6] T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J. Stankovic, T. Abdelzaher, J. Hui, and B. Krogh, "VigilNet: An integrated sensor network system for energy-efficient surveillance," *ACM Transactions on Sensor Networks*, vol. 2, no. 1, pp. 1–38, 2006.

[7] C.-M. Hung, J.-J. Chen, and T.-W. Kuo, "Energy-efficient real-time task scheduling for a DVS system with a non-DVS processing element," in *Proceedings of the Real-Time Systems Symposium*, Dec. 2006, pp. 303–312.

[8] Website, 2007, http://www.intel.com/design/mobile/datashts/309221.htm.

[9] R. Jejurikar, C. Pereira, and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," in *Proceedings of the Design Automation Conference*, Jun. 2004, pp. 275–280.

[10] E. Jensen, C. Locke, and H. Tokuda, "A time-driven scheduling model for real-time operating systems," in *Proceedings of the Real-Time Systems Symposium*, Dec. 1985, pp. 112–122.

[11] W.-C. Kwon and T. Kim, "Optimal voltage allocation techniques for dynamically variable voltage processors," *ACM Transactions on Embedded Computing Systems*, vol. 4, no. 1, pp. 211–230, Feb. 2005.

[12] H. Li, P. Shenoy, and K. Ramamritham, "Scheduling messages with deadlines in multi-hop real-time sensor networks," in *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, Mar. 2005, pp. 415–425.

[13] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[14] A. Mainwaring, J. Polastre, R. S. D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proceedings of the International Workshop on Wireless Sensor Networks and Applications*, Sep. 2002, pp. 88–97.

[15] B. Mochocki, D. Rajan, X. Hu, C. Poellabauer, K. Otten, and T. Chantem, "Network-aware dynamic voltage and frequency scaling," in *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, Apr. 2007, pp. 215–224.

[16] A. Mok, X. Feng, and D. Chen, "Resource partition for real-time systems," in *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, May 2001, pp. 75–84.

[17] L. Niu and G. Quan, "Reducing both dynamic and leakage energy consumption for hard real-time systems," in *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, Sep. 2004, pp. 140–148.

[18] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *ACM Operating Systems Review*, vol. 35, no. 5, pp. 89–102, Oct. 2001.

[19] C. Poellabauer and K. Schwan, "Energy-aware traffic shaping for wireless real-time applications," in *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, May 2004, pp. 48–55.

[20] K. Shin and C. Hou, "Design and evaluation of effective load sharing in distributed real-time systems," *IEEE Transactions on Parallel & Distributed Systems*, vol. 5, no. 7, pp. 704–719, Jul. 1994.

[21] E. Uysal-Biyikoglu, B. Prabhakar, and A. E. Gamal, "Energy-efficient packet transmission over a wireless link," *IEEE Transactions on Networking*, vol. 10, no. 4, pp. 487–499, Aug. 2002.

[22] F. Wang, K. Ramamrithnam, and J. Stankovic, "Determining redundancy levels for fault-tolerant real-time systems," *IEEE Transactions on Computers*, vol. 44, no. 2, pp. 292–301, Feb. 1995.

[23] R. West and C. Poellabauer, "Analysis of a window-constrained scheduler for real-time and best-effort packet streams," in *Proceedings of the Real-Time Systems Symposium*, Dec. 2000, pp. 239–248.

[24] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proceedings of the Symposium on Foundations of Computer Science*, Oct. 1995, pp. 374–382.

[25] J. Yi, C. Poellabauer, X. Hu, J. Simmer, and L. Zhang, "Energy-conscious co-scheduling of tasks and packets in wireless real-time environments," in *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, Apr. 2009, pp. 265–274.