An Energy-Efficient Slack Distribution Technique for Multimode Distributed Real-Time Embedded Systems

Rabi N. Mahapatra, Senior Member, IEEE, and Wei Zhao, Fellow, IEEE

Abstract—In multimode distributed systems, active task sets are assigned to their distributed components for realizing one or more functions. Many of these systems encounter runtime task variations at the input and across the system while processing their tasks in real time. Very few efforts have been made to address energy efficient scheduling in these types of distributed systems. In this paper, we propose an analytical model for energy efficient scheduling in distributed real-time embedded systems to handle time-varying task inputs. A new slack distribution scheme is introduced and adopted during the schedule of the task sets in the system. The slack distribution is made according to the service demand at the nodes which affects the energy consumption in the system. The active component at a node periodically determines the service rate and applies voltage scaling according to the dynamic traffic condition observed at various network nodes. The proposed approach uses a comprehensive traffic description function at nodes and provides adequate information about the worst-case traffic behavior anywhere in the distributed network, thereby enhancing the system power management capabilities. We evaluate the proposed technique using several benchmarks employing an event driven simulator and demonstrate its performance for multimode applications. Experimental results indicate significant energy savings in various examples and case studies.

Index Terms—Multimode, traffic descriptor, slack management, service rate, low-power.

1 INTRODUCTION

Many embedded command and control systems used in manufacturing, chemical processing, avionics, telemedicine, and sensor networks are mission-critical. These systems usually comprise of applications that must accomplish certain functionalities in real-time. These functions subsist as realizable task sets and they use system components to process the data/messages associated with the task set. Each task set is represented as a chain of tasks mapped onto the distributed system components. Consequently, messages of a task set have deadlines by which they must be processed. The set of tasks currently active in the system (active task set) is determined by the set of applications currently running in the system. Such distributed embedded systems tend to operate in a modal fashion while sharing system resources. Therefore, the set of applications in the system can vary with time. As a result, the active task sets in the system also vary from time to time. In this paper, we propose a technique which enables such systems to cope with these task variations and guarantee end-to-end and local deadlines of messages while minimizing energy consumption.

There has been an extensive study on low power scheduling of periodic tasks, aperiodic tasks, and their combinations on single processors as in [3], [4], [5], [6]. However, relatively little work has been done in the area of low-power scheduling for multiprocessors [7], [8], [9], [10], [11], [12], [13], [14], [15], [16]. In [6], authors assume the tasks to be independent and known a priori. Power conscious algorithms for joint scheduling of periodic task graphs and aperiodic tasks are proposed in [7], [15], [16]. The work in [8] performs joint scheduling of distributed periodic and hard aperiodic tasks by determining the static schedule that meets timing and synchronization requirements. The work in [9] presents a scheduling discipline called Elastic Round Robin (ERR), which allows fair and efficient scheduling of packets to improve the isolation between different users. A power-cost localized routing algorithm that attempts to minimize the total power needed to route a message between a source and a destination is proposed in [10]. This algorithm tries to minimize the total power by avoiding nodes with little remaining battery power. The work in [11] proposes static and dynamic power management schemes for a set of real-time tasks with precedence constraints. The work in [12] proposes a method for the design of communication-based power management systems in which the system level architecture regulates the execution of various system components with the aim of improving battery life.

The above approaches target the systems with fixed active tasks. For distributed embedded systems, the data communication time also varies significantly within the network due to resource sharing and variations in processing delays at the components. Most of the existing distributed systems usually do not take these variations into account and, therefore, may not explore the slacks...
available to further reduce the energy consumption. The proposed approach is flexible in a way to accommodate these variations and is applicable to many distributed embedded systems. In addition, the proposed approach has no restrictions on task sets whether their deadlines need to be less than or equal to their period.

The task sets are assigned to the system components to process incoming data in order to guarantee their deadlines. These deadlines could be local (i.e., deadline of a computational task on a particular node) or end-to-end (i.e., the total delay suffered by the messages of a task set through various nodes or components). According to the decomposition approach described in [2], the worst-case end-to-end delays can be computed by summing up the upper bounds on the delays suffered by the messages of a task set at each node involved in the processing. A key issue in guaranteeing that the deadlines are satisfied is the derivation of an upper bound on the delays suffered by the messages of a task set. However, admitting a new task set perturbs the traffic of some of the existing task sets, necessitating the re-evaluation of the end-to-end message delays of these task sets. To analyze the delay bounds, we use a maximum demand function to describe a task set’s traffic at the input of each node. The additional benefit of using the demand function is that it does not mandate uniform message processing time. Since the scheduling policy used at each node determines the order in which the messages are processed, it will have a direct impact on the worst-case delay experienced by the messages of a task set.

In this paper, we consider two widely used scheduling policies: First-Come-First-Serve (FCFS) and Weighted-Round-Robin (WRR). We exploit the fact that the messages of a task set at the input of each node determine the order in which the messages are processed, it will have a direct impact on the worst-case delay experienced by the messages of a task set.

In this paper, we consider two widely used scheduling policies: First-Come-First-Serve (FCFS) and Weighted-Round-Robin (WRR). We exploit the fact that the messages of a task set at the input of each node determine the order in which the messages are processed, it will have a direct impact on the worst-case delay experienced by the messages of a task set.

In this paper, we consider two widely used scheduling policies: First-Come-First-Serve (FCFS) and Weighted-Round-Robin (WRR). We exploit the fact that the messages of a task set at the input of each node determine the order in which the messages are processed, it will have a direct impact on the worst-case delay experienced by the messages of a task set.

In this paper, we consider two widely used scheduling policies: First-Come-First-Serve (FCFS) and Weighted-Round-Robin (WRR). We exploit the fact that the messages of a task set at the input of each node determine the order in which the messages are processed, it will have a direct impact on the worst-case delay experienced by the messages of a task set.

In this paper, we consider two widely used scheduling policies: First-Come-First-Serve (FCFS) and Weighted-Round-Robin (WRR). We exploit the fact that the messages of a task set at the input of each node determine the order in which the messages are processed, it will have a direct impact on the worst-case delay experienced by the messages of a task set.

In this paper, we consider two widely used scheduling policies: First-Come-First-Serve (FCFS) and Weighted-Round-Robin (WRR). We exploit the fact that the messages of a task set at the input of each node determine the order in which the messages are processed, it will have a direct impact on the worst-case delay experienced by the messages of a task set.

In this paper, we consider two widely used scheduling policies: First-Come-First-Serve (FCFS) and Weighted-Round-Robin (WRR). We exploit the fact that the messages of a task set at the input of each node determine the order in which the messages are processed, it will have a direct impact on the worst-case delay experienced by the messages of a task set.

In this paper, we consider two widely used scheduling policies: First-Come-First-Serve (FCFS) and Weighted-Round-Robin (WRR). We exploit the fact that the messages of a task set at the input of each node determine the order in which the messages are processed, it will have a direct impact on the worst-case delay experienced by the messages of a task set.

In this paper, we consider two widely used scheduling policies: First-Come-First-Serve (FCFS) and Weighted-Round-Robin (WRR). We exploit the fact that the messages of a task set at the input of each node determine the order in which the messages are processed, it will have a direct impact on the worst-case delay experienced by the messages of a task set.

In this paper, we consider two widely used scheduling policies: First-Come-First-Serve (FCFS) and Weighted-Round-Robin (WRR). We exploit the fact that the messages of a task set at the input of each node determine the order in which the messages are processed, it will have a direct impact on the worst-case delay experienced by the messages of a task set.

In this paper, we consider two widely used scheduling policies: First-Come-First-Serve (FCFS) and Weighted-Round-Robin (WRR). We exploit the fact that the messages of a task set at the input of each node determine the order in which the messages are processed, it will have a direct impact on the worst-case delay experienced by the messages of a task set.

In this paper, we consider two widely used scheduling policies: First-Come-First-Serve (FCFS) and Weighted-Round-Robin (WRR). We exploit the fact that the messages of a task set at the input of each node determine the order in which the messages are processed, it will have a direct impact on the worst-case delay experienced by the messages of a task set.

In this paper, we consider two widely used scheduling policies: First-Come-First-Serve (FCFS) and Weighted-Round-Robin (WRR). We exploit the fact that the messages of a task set at the input of each node determine the order in which the messages are processed, it will have a direct impact on the worst-case delay experienced by the messages of a task set.

In this paper, we consider two widely used scheduling policies: First-Come-First-Serve (FCFS) and Weighted-Round-Robin (WRR). We exploit the fact that the messages of a task set at the input of each node determine the order in which the messages are processed, it will have a direct impact on the worst-case delay experienced by the messages of a task set.
that interval. Each processing node in the network is assumed to be voltage/frequency scalable and has a maximum service rate specified by $\sigma_{\text{max}}$. The instantaneous processing time demand function $\Gamma_i$ is the traffic descriptor at that node. The mathematical derivation for “worst case delay” at a particular node and the “traffic descriptor” details are discussed in [1]. The proposed technique and its analysis are based on those initial studies.

2.2 Motivational Example

Example 1. Let us consider a simple distributed embedded system as in Fig. 1 that consists of three components/nodes (PE$_1$, PE$_2$, PE$_3$) and two task sets ($\prod_1$, $\prod_2$). Each task set realizes a function that is part of some desired application. Function 1 involves the processing of task set $\prod_1$ served by nodes PE$_1$, PE$_3$, and PE$_2$ as indicated with a solid line. Function 2 corresponds to the task set $\prod_2$, which is executed at nodes PE$_1$ and PE$_2$. An application consists of a function or a combination of some of these functions that illustrate various modes of a multifunctional system. Thus, there are three possible modes ($\prod_1$, $\prod_2$, $\prod_1 + \prod_2$) of operation in this system. Table 1 lists the specification of task set $\prod_1$, when served by different nodes.

Let us consider the node PE$_3$ for simplicity. (The techniques to deal with more complex scenarios involving several connections are presented in Section 3). Let $f'$ be the normal frequency of operation in this processor. From specification, the maximum processing time, $\Gamma(I)$, demanded by the messages during an interval ($I = 9$ seconds) at node PE$_3$ is 5 seconds. Further, the messages suffer a worst-case delay of 5 seconds at the node PE$_3$. A task set can be admitted to the system if the end-to-end worst-case delay at the random nodes is less than the end-to-end deadline. Here, $\prod_1$ can be admitted into the system due to its worst-case delay (25 + 35 + 5) 65 seconds is less than its end to end deadline 95 seconds.

Example 2. Let us consider the node PE$_3$ again from the above example. Table 2 shows the runtime variations in the actual processing time demanded by the messages of the task set $\prod_1$ at node PE$_3$ during successive monitoring intervals. The variation in processing time demand is due to the traffic fluctuations caused by varying input tasks, idle intervals and over estimation of incoming messages based on previous intervals. We take advantage of these runtime variations to reduce the energy consumption at various nodes. We will determine the service rate of a node that is needed to guarantee the processing of the messages by their worst-case delay $d$, at the beginning of each interval. At the beginning of the first interval $[t_0, t_0 + I]$, the clock speed is set to $0.33f$ as determined from Example 1.

At the beginning of the second interval, we knew that the actual computation time demanded was 4 seconds and the processor was able to serve, $(\Gamma(I)/d) \times I$ (i.e., $(5/15) \times 9 = 3$ sec), with reference to the normal clock speed. The remaining 1 second of computational demand must be processed within $(d - I)$ time units, i.e., 6 seconds. The PE$_3$ must be able to serve this traffic and the expected processing time demand by the messages of the upcoming interval, i.e., 9, by their delays. Hence, the required operational frequency is set at $0.46f (1/6 + 5/15)$. In practical systems, however, the clock frequency cannot be scaled continuously. So, the processor will be put in the lowest suitable frequency mode higher than this factor. Fig. 2 shows the clock speed of the original schedule and new schedule over four consecutive intervals. By reducing the frequency of operation at a processing node, voltage scaling can be applied to reduce the power consumption.

---

**TABLE 1**

Traffic Specification for an Arbitrary Interval of Length $I = 9$ Seconds

<table>
<thead>
<tr>
<th>Nodes</th>
<th>$\Gamma(I)$</th>
<th>Worst-case delay</th>
<th>End-to-end deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE$_1$</td>
<td>6</td>
<td>25</td>
<td>95</td>
</tr>
<tr>
<td>PE$_2$</td>
<td>8</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>PE$_3$</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

---

**TABLE 2**

Runtime Variations in Processing Time Demands of $\prod_1$ at PE$_3$

<table>
<thead>
<tr>
<th>Interval</th>
<th>Actual Computation time demanded</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[t_0, t_0 + I]$</td>
<td>4 sec</td>
</tr>
<tr>
<td>$[t_0 + I, t_0 + 2I]$</td>
<td>4 sec</td>
</tr>
<tr>
<td>$[t_0 + 2I, t_0 + 3I]$</td>
<td>2 sec</td>
</tr>
</tbody>
</table>
2.3 Derivation of Maximum Demand Function

As shown in Fig. 1, any arbitrary task set \( \prod_i \) can be mapped onto a sequence of processing nodes. Let us represent the sequence of processing nodes serving task set \( \prod_i \) by \( s(i, 1), s(i, 2), \ldots, s(i, l_i) \), where \( l_i \) is the total number of nodes serving task set \( \prod_i \) and \( s(i, j) \) denotes the node-id of the \( j \)th node. Let us assume that \( d_i, s(i, 1), d_i, s(i, 2), \ldots, d_i, s(i, k) \) are the upper bounds on the delays suffered by \( \prod_i \) at the first \( k (k < l_i) \) nodes serving the task set. Then, the upper bounds on \( \Gamma_{i,s(i,k+1)}(I) \), the maximum demand function for the task set \( \prod_i \) at the output node \( s(i, k) \), is given by:

\[
\Gamma_{i,s(i,k+1)}(I) \leq \min \left( C_{i,k+1}, \left( 1 + \frac{y^k_T}{T} \right) \Gamma_{i,s(i,1)}(I + y^k_T) \right), \tag{1}
\]

where \( y^k_T \) is the sum of the upper bounds on the delays experienced by \( \prod_i \)'s messages at all the upstream nodes from \( s(i, 1) \) to \( s(i, k) \) and is given by:

\[
y^k_T = \sum_{l \leq k} d_{i,s(i,l)}. \tag{2}
\]

The upper bound on the maximum demand function \( \Gamma(I) \) at any particular node is obtained by summing the upper bounds on the maximum demand function of all the incoming task sets, i.e.,

\[
\Gamma(I) = \sum_{i=1}^r \Gamma(i), \tag{3}
\]

where \( r \) is the maximum number of task sets entering a particular node.

2.4 Delay Analysis

In this section, we derive the upper bounds on delays due to messages of a task set at any arbitrary node in the network of components. We consider the First-Come-First-Serve (FCFS) and the Weighted-Round-Robin (WRR) scheduling policies along with multiple task sets served at that node. Let \( \{\prod_{i,1}, \ldots, \prod_{i,n}\} \) be the incoming task sets being served at any arbitrary node “\( j \)” with corresponding traffic descriptors \( (\Gamma^1, \Gamma^2, \ldots, \Gamma^i, \ldots, \Gamma^R) \).

2.4.1 FCFS (First-Come-First-Serve)

With the FCFS scheduling policy, messages from every incoming task set are placed in a common queue. Every message is delayed by the time required for the node to process the messages ahead of it in the queue. Hence, the maximum delay experienced by any message occurs when it is the last message in the queue and the queue length is at its maximum. The following theorem gives the maximum delay experienced by the messages of a task set with FCFS policy at any arbitrary node.

**Theorem 1.** For an FCFS Server, the maximum delay suffered by the messages of task set \( \prod_i \) at any arbitrary node is given by:

\[
d^\text{FCFS}_i = \max_{0 \leq T \leq B} \left( \sum_{i=1}^l \Gamma^i(T) - T \right), \tag{4}
\]

where \( B \) is the busy interval and is given by

\[
B = \min \left\{ T | \sum_{i=1}^l \Gamma^i(T) \leq T \right\}.
\]

**Proof.** With FCFS scheduling, it is easy to see that the length of the busy interval for any arbitrary task set is \( B = \min\{T | \sum_{i=1}^l \Gamma^i(T) \leq T \} \). Further, the initial size of queue is zero at the beginning of a busy interval. The total processing time demanded by the messages of incoming task sets from the starting of a busy interval over a period of length \( T \) is given by \( \sum_{i=1}^l \Gamma^i(T) \). The total time spent by the node serving the messages from this task set during this interval is \( T \). Since the size of the queue at the beginning of the busy interval is zero, the length of the queue at the end of the interval \( T \) is given by \( \sum_{i=1}^l (\Gamma^i(T) - T) \). With FCFS scheduling, the worst-case delay suffered by the messages of a task set is the maximum queue length and is given by

\[
d^\text{FCFS}_i = \max_{0 \leq T \leq B} \left( \sum_{i=1}^l \Gamma^i(T) - T \right). \tag{5}
\]

2.4.2 WRR (Weighted-Round-Robin)

In a WRR server, each task served at an arbitrary node is assigned a separate buffer. Further, each task set \( \prod_i \) is assigned a weight \( w_i \). The WRR server ensures that the task set \( \prod_i \) receives a minimum service rate of \( w_i / F \), where \( F = \sum_{i=1}^l w_i \). In a WRR server, if the queue for task set \( \prod_i \) is
Theorem 3. During the duration of length $T$, if the messages from task set $\prod_i$ are processed in the interval $T$ for the duration of length, $S_{WRR}^{i,j}(T) = \left\lceil \frac{T}{F} \right\rceil w_i + \max \left(0, T - \left\lceil \frac{T}{F} \right\rceil F - F + w_i \right).$ (6)

Theorem 2. For a WRR server, if the queue for task set $\prod_i$ is not empty during a time interval of length $T$, then the maximum queue size for the task set $\prod_i$ at any given node is given by

$$Q_i = \max_{0 \leq T \leq B} \left( \Gamma'(T) - S_{WRR}^{i,j}(T) \right),$$

where $B = \min(T) \left[ \Gamma'(T) \leq S_{WRR}^{i,j}(T) \right]$.

Proof. Theorem 2 can be proved using an argument similar to the one used in Theorem 1. Using this theorem, we now derive the worst-case delay of a message from task set $\prod_i$. □

Theorem 3. For a WRR Server, the maximum delay for task set $\prod_i$, at any arbitrary node is given by

$$d_{WRR}^{i,j} = \left\lceil \frac{Q_i}{w_i} \right\rceil (F - w_i) + Q_i.$$

Proof. For task set $\prod_i$, the worst-case delay is experienced by the last message in the buffer when the length of the queue reaches $Q_i$, its maximum value. The waiting time of this last message can be considered to have two components: The time during which node process task sets other than $\prod_i$ and the time when the node transmits messages of $\prod_i$ that are ahead of the last message. Clearly, the node requires $\left\lceil \frac{Q_i}{w_i} \right\rceil$ cycles to transmit the last message in the queue. In the worst case, each cycle is of length $F$ and $\prod_i$ is served last in each cycle. Consequently, task set $\prod_i$ does not receive any service during $(F - w_i)$ portion of every cycle of length $F$. Therefore, the worst case waiting time for $\prod_i$ is $\left\lceil \frac{Q_i}{w_i} \right\rceil (F - w_i)$. The time to transmit $Q_i$ messages is $Q_i$. Hence, the theorem follows. □

2.5 Mechanism to Admit New Task Sets

The proposed system intends to handle tasks with variable arrival time similar to scenarios seen in some integrated multimedia applications. There is no restriction requiring the periodic tasks deadline to be less or equal to their period of arrival. Without such restrictions, more instances of a task set may arrive at the system for processing and will cause network congestion. If the new tasks are accepted without admittance test, it will lead to the system’s eventual failure.

The technique for admitting new task sets to a distributed real-time system is presented here. Consider a system that has already admitted a set of “n” hard real-time task sets $\{\prod_1, \ldots, \prod_n\}$. This implies that the messages due to these task sets can be processed by their deadlines. When requests from a new task set $\prod_{n+1}$ arrives for admission, the system must determine whether the messages in $\prod_{n+1}$ task set can be processed by their deadlines without violating the guaranteed deadlines of the existing task sets $\{\prod_1, \ldots, \prod_n\}$. Usually, the admittance of a new task set perturbs the traffic of some of the existing task sets and also affects the worst-case delays suffered by these task sets. Such interdependencies among task sets are identified by the use of connection-server graphs as described in [1]. A new task set $\prod_{n+1}$ will be admitted if it satisfies the following three constraints:

1. The local deadlines at each computational node are satisfied, i.e., the upper bound on the worst-case delay suffered by a task set at a particular node is bounded by its local deadline at that node.
2. The sum of the worst-case delays, $\sum_{i,j} d_{i,j}$, suffered by the new task set at the corresponding nodes on the network is less than the end-to-end deadline, $D_{in}$.
3. The recalculated worst-case delays of the messages of affected task sets are bounded by their deadlines. The task set server graph is used to identify the affected task sets.

The notations in Table 3 describe the system parameters at any arbitrary node $s_{ij}$ involved in serving the task set $\prod_i$.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t.s.rate</td>
<td>sum of the service rates of the first $j$ nodes involved in the task set $\prod_i$.</td>
</tr>
<tr>
<td>s.rate$_i$</td>
<td>service rate for task set $\prod_i$ at node $s_{ij}$.</td>
</tr>
<tr>
<td>t.delay</td>
<td>sum of the worst-case-delay of the messages of $\prod_i$ at the first $j$ nodes.</td>
</tr>
<tr>
<td>$n$</td>
<td>total number of incoming task sets at the node $s_{ij}$, for a given interval of length $I$.</td>
</tr>
<tr>
<td>slack$_i$</td>
<td>slack allocated to node $s_{ij}$ due to task set $\prod_i$.</td>
</tr>
<tr>
<td>t.slack$_{(1...j)}$</td>
<td>total slack to be distributed among first ‘$j’ nodes serving the task set $\prod_i$.</td>
</tr>
<tr>
<td>wc.delay$_i$</td>
<td>Worst case delay for the task set $\prod_i$ at node $s_{ij}$.</td>
</tr>
<tr>
<td>wrr.delay</td>
<td>Worst case delay for task set $\prod_i$, at $s_{ij}$ when WRR policy is applied.</td>
</tr>
<tr>
<td>fcfs.delay</td>
<td>Worst case delay for task set $\prod_i$, at $s_{ij}$ when the FCFS policy is applied.</td>
</tr>
</tbody>
</table>

2.5 Mechanism to Admit New Task Sets

The proposed system intends to handle tasks with variable arrival time similar to scenarios seen in some integrated multimedia applications. There is no restriction requiring the periodic tasks deadline to be less or equal to their period of arrival. Without such restrictions, more instances of a task set may arrive at the system for processing and will cause network congestion. If the new tasks are accepted without admittance test, it will lead to the system’s eventual failure.

The technique for admitting new task sets to a distributed real-time system is presented here. Consider a system that has already admitted a set of “n” hard real-time task sets $\{\prod_1, \ldots, \prod_n\}$. This implies that the messages due to these task sets can be processed by their deadlines.

When requests from a new task set $\prod_{n+1}$ arrives for admission, the system must determine whether the messages in $\prod_{n+1}$ task set can be processed by their deadlines without violating the guaranteed deadlines of the existing task sets $\{\prod_1, \ldots, \prod_n\}$. Usually, the admittance of a new task set perturbs the traffic of some of the existing task sets and also affects the worst-case delays suffered by these task sets. Such interdependencies among task sets are identified by the use of connection-server graphs as described in [1]. A new task set $\prod_{n+1}$ will be admitted if it satisfies the following three constraints:

1. The local deadlines at each computational node are satisfied, i.e., the upper bound on the worst-case delay suffered by a task set at a particular node is bounded by its local deadline at that node.
2. The sum of the worst-case delays, $\sum_{i,j} d_{i,j}$, suffered by the new task set at the corresponding nodes on the network is less than the end-to-end deadline, $D_{in}$.
3. The recalculated worst-case delays of the messages of affected task sets are bounded by their deadlines. The task set server graph is used to identify the affected task sets.

The notations in Table 3 describe the system parameters at any arbitrary node $s_{ij}$ involved in serving the task set $\prod_i$. The Algorithm in Fig. 3 for task set admittance has two major functions. 1) Using Procedure “task set_request,” it examines incoming requests from task sets at admission time, and 2) with the help of procedure “task set_reply,” it notifies if the incoming task set was accepted or rejected depending on the status of the existing task set and their deadline constraints. The task sets are admitted into the

<table>
<thead>
<tr>
<th>Table 3 List of Notations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notation</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td>t.s.rate</td>
</tr>
<tr>
<td>s.rate$_i$</td>
</tr>
<tr>
<td>t.delay</td>
</tr>
<tr>
<td>$n$</td>
</tr>
<tr>
<td>slack$_i$</td>
</tr>
<tr>
<td>t.slack$_{(1...j)}$</td>
</tr>
<tr>
<td>wc.delay$_i$</td>
</tr>
<tr>
<td>wrr.delay</td>
</tr>
<tr>
<td>fcfs.delay</td>
</tr>
</tbody>
</table>
system in a first come first serve basis. Since all the task sets are of hard real-time, their local and end-to-end deadlines specifications are met at all the nodes before they are admitted. The worst case delay value, “wc_delay” is calculated at each node involved to process the task set to estimate the end-to-end delay. The end-to-end delay is then compared with the hard deadline specified by the function. If it meets the deadline, then the task is admitted into the distributed system else rejected.

3 LOW POWER TECHNIQUE

In this section, we present elaborate discussions on the proposed low-power technique. This technique has two components:

- Section 3.1 shows how we maximize the use of available slack in processing the task set.
- Section 3.2 shows how we handle runtime variations in the execution time, idle intervals, and nonarrivals.

3.1 Slack Distribution

Any arbitrary task set $\prod_i$ is processed through a sequence of nodes. Let us represent the sequence of nodes that serve the task set $\prod_i$ as $s(i, 1), s(i, 2), \ldots, s(i, j), \ldots s(i, k)$, where $k$ is the total number of nodes serving the task set. The upper bounds on the delays incurred by the messages of the task set $\prod_i$ at various nodes are represented by the set $\{d_{i, 1}, d_{i, 2}, \ldots, d_{i, j}, \ldots d_{i, k}\}$. $D_{i,k}$ denotes the end-to-end deadline for the task set $\prod_i$. The total slack available in the task set is given by the equation:

$$t_{slack}(1, \ldots, j, \ldots, k) = D_{i,k} - \sum_{j=1}^{k} d_{i,j}. \quad (9)$$

We propose a slack distribution algorithm to distribute this slack amongst nodes proportional to their service rates. The service rate determination details are given in Section 4. Since the task sets are allowed to arrive at the system dynamically, their admission is handled by the task set establishment phase. A task set is established only if the following criteria are satisfied:

1. The worst-case delay suffered by the messages of the incoming task set at various nodes is less than the corresponding local task deadlines.
2. The sum of the worst-case delays along the involved nodes is less than the end-to-end deadline for the incoming task set.
3. The recalculated worst-case delays of the existing task sets in the system are bounded by their specified deadlines.

One of the key ideas behind the proposed low power technique is that the processing of messages of incoming task sets at any arbitrary node can be extended up to the upper bound of their worst-case delays. We reduce the service rate at nodes by using frequency/voltage scaling to the maximum possible extent such that their processing will be completed by their delay bounds. Hence, the larger the upper bounds on the delays, the smaller the service rate and greater the power savings. These worst-case delays at individual nodes can be increased by effectively distributing the slack. In the proposed approach, the slack is allocated to a node proportional to the ratio of the service rate at the node and the sum of the service rates of all the nodes serving the task set. Furthermore, the proportion of the slack distributed and the worst-case delay experienced by the messages of a task set at a particular node are bounded by the local deadline at that node.

3.1.1 First-Come-First-Serve (FCFS) Scheduling

With FCFS Scheduling policy, the task set with minimum available slack places an upper bound on the amount of delays by which the worst-case delay at a node can be increased. Hence, there is no benefit in allocating higher slack at that node for other task sets. The algorithm in Fig. 4...
to determine the service rate of the current interval. The new service rate should satisfy the following two conditions.

- First, it should guarantee the processing of the messages in the upcoming interval by their delay bounds.
- Second, this service rate must guarantee the processing of the messages that were left in the queue (arrived during the previous intervals) without being serviced by their delay bounds.

The periodic service rate calculation takes place at every node at the beginning of every monitoring interval and it depends on the scheduling policy at the processing nodes. The following definitions are used to describe the system parameters:

- \( I \) represents the length or duration of monitoring interval at a particular node. It is calculated by dividing the \( wc_{ delay} \) by \( z \), where \( z \) is the number of monitoring intervals.
- \( \omega_I \) represents the fraction of the interval \( I \) during which the messages from task set \( \Pi_i \) will be processed.
- \( wc_{ delay}^i \) represents the worst case delay suffered by the messages from task set \( \Pi_i \) at given node \( s_{ ij} \) serving the task set.
- \( t_{ start} \) gives the system start time. The queue content is zero at this time.
- \( p^i_I \) represents the actual processing time demanded by the messages of task set \( \Pi_i \) that have already arrived at the node before the time instant \( t \).
- \( q^i_I \) represents the processing time demanded by the unprocessed messages left in the queue by task set \( \Pi_i \) which arrived during the interval \( (t - (rI) - (t - (r - 1)I)) \) at time instant \( t \) where \( r \) is the maximum number of task sets entering a particular node.
- \( s_{ rate}^i_{tstart} \) represents the required service rate at time instant \( t \) to guarantee the processing of the messages from task set \( \Pi_i \) in the queue which arrived during the interval \( (t - (rI) - (t - (r - 1)I)) \) by their worst case delay bounds.
- \( s_{ rate}^i \sum_{r=1}^z s_{ rate}^i_{tstart} \) represents the required service rate at the node at a time \( t \) to process the messages from task set \( \Pi_i \).

Suppose \( \{\Pi_1, \ldots, \Pi_k\} \) represent the incoming task sets at a given node and \( \{\Gamma^1, \Gamma^2, \ldots, \Gamma^i, \ldots, \Gamma^k\} \) be the corresponding input traffic descriptor set. Let \( \Gamma_I \) be the input traffic descriptor function for task set \( \Pi_i \) at time \( t \).

At the system start time, the service rate is set as follows:

\[
s_{ rate}^i_{tstart} \geq \Gamma^i_{tstart}(I)/\omega_{wc_{ delay}}.
\]  

This service rate guarantees the processing of the messages of task set \( \Pi_i \) that will arrive in the upcoming interval by their delay bounds. At the start of a new monitoring interval, a new service rate value is computed. This new service rate shall guarantee the processing of the messages that will arrive in the upcoming interval, as well as the messages that already arrived during previous intervals and have not been processed, by their delay bounds. The
new service rate at the beginning of every interval is determined according to the following equation

\[ s_{rate}^i_t = \sum_{r=1}^{n} s_{rate}^i_{r,t} + \gamma^i_t(I)/\omega^i_{we-delay}_t, \quad (11) \]

and the corresponding queue is determined according to

\[ q^i_t = \sum_{r=1}^{n} q^i_{r,t}, \quad (12) \]

where \( u = \{t - \max\{t_{start}, t - (z - 1)I\}/I\}. \)

The service rate \( s_{rate}^i_{r,t} \) and the corresponding processing time demanded by the outstanding messages that arrived during the interval \((t - (rI), (t - (r - 1)I))\) are given by

\[ s_{rate}^i_{r,t} \times (we-delay_t - rI) = q^i_{r,t}, \quad (13) \]

\[ q^i_{r,t} = \left\{ \begin{array}{ll}
\chi & \psi > 0 \\
\chi + \psi & o.w.,
\end{array} \right. \quad (14) \]

where

\[ \chi = p^i_{t-rI} - p^i_{t-(r-1)I}, \quad (15) \]

and

\[ \psi = \left( q^i_{r,t} - \sum_{u=1}^{r} \int_{t-(u-1)I}^{t} s_{rate}^i_{r-uI} \right) > 0. \quad (16) \]

**Theorem 4.** The service rate, \( s_{rate}^i_t \), will guarantee the processing of the messages of task set \( T_i \), by their worst case delay bounds.

**Proof.** To prove that \( s_{rate}^i_t \) is a valid service rate to guarantee the end-to-end deadline of a message of a task set, two cases have to be proven satisfactorily:

**Case 1.** The unprocessed messages that arrived during any outstanding previous interval \((t - rI, t - (r - 1)I)\) will have to be processed within the upper bound on their delays by the service rate \( s_{rate}^i_t \), i.e.,

\[ s_{rate}^i_{r,t}(we-delay_t - rI) \geq q^i_{r,t} \]

\[ (s_{rate}^i_{r,t} + \ldots + s_{rate}^i_{r,t}) (we-delay_t - rI) \geq q^i_{r,t} \]

\[ (s_{rate}^i_{r,t} + \ldots + s_{rate}^i_{r,t}) (we-delay_t - rI) \geq q^i_{r,t} \]

Substituting from (8), which is valid since 

\[ s_{rate}^i_{u,t} \times (d - rI), \geq 0 \forall \ u. \]

This proves the theorem.

**Case 2.** The messages that will arrive during the upcoming interval and the messages that are in the queue will also be processed within their deadlines using the service rate. In other words,

\[ s_{rate}^i_{u,t} \times (d - rI), \geq 0 \forall \ u. \]

which is valid since \( rI \geq 0 \forall \ u. \) This proves the theorem. \( \square \)

The proof for FCFS scheduling policy will follow the above steps. The only difference is in the way worst-case delay is calculated. For FCFS, there is only a single queue, which implies that there is only one worst-case delay value for all the task sets in a given mode of operation. In contrast, for WRR, there are several queues at a node, one corresponding to each task set.

The overhead of the periodic service rate is determined by the scheduling policy at a given node. For FCFS, it is \( O(I) \). For WRR it is \( O(II) \), where \( l \) is the total number of task sets being served at a node in a particular time interval and \( I \) is an integer.

### 4 Experiments and Results

In this section, we present relevant experiments, results and a case study to demonstrate the benefits of the proposed slack distribution technique in reducing energy consumption.

#### 4.1 Experimental Setup

We developed an event driven simulator and used an actual embedded processor as one of the node in our cosimulation experimental environment to validate the proposed techniques. The simulator runs on a 2GHz processor with Linux OS. In order to estimate the periodic service rate overhead, we employ an Intel PXA250 XScale embedded processor as the processing node of the distributed framework. The XScale embedded processor is a dynamic voltage scalable processor with four different frequency setups for energy saving. The system-level energy savings were estimated using the specifications of the XScale processor [17]. The test benches consist of three synthetic and two real world test cases as given in Table 4. The integrated multimedia test case is a multifunctional application consisting of MPEG, JPEG, MP3, and ADPCM task sets operating in three different modes. Each mode is a combination of these functions (Mode 1: MPEG, MP3; Mode 2: MPEG, JPEG, ADPCM; Mode 3: all the four applications). The DSP application consists of small functions that only operate in a single mode [18].

#### 4.2 Results and Discussions

When more task sets are allowed into the distributed system, the service rates at the nodes have to be increased to meet the application deadlines. To increase the energy saving, the service rate must decrease at nodes by exploiting the available slacks in the system. This decrease in service rate consequently limits the admission of task sets resulting in a drop in system inputs. Thus, there is a trade-off between the rate of dropping the task sets and overall service rate. For various test cases, we vary the service rate from 50 to 100 percent in our experiment as shown in Fig. 5.
We notice that the task set drop rate falls when the service rate is increased at the cost of more energy consumption. The choice of WRR scheduling scheme at the nodes has been randomly chosen in this experiment with multimode operations. A higher slope on drop rate is seen in the case of integrated multimedia task sets. For the integrated multimedia case, the drop rate is locally insensitive to the increase in service rate between the large slopes. This is due to the high granularity of task sets in integrated multimedia test cases compared to the other benchmarks. A comparison of various slack distribution schemes is depicted in Fig. 6. The proposed slack distribution technique $S_{rate}$ outperforms the other three known slack distribution schemes (Greedy, Equal, and Worst-case-execution time).

This indicates that reducing the service rate of a node with higher utilization factor will have a greater impact on system-wide energy saving than reducing the service rate of a node with lower utilization by the same factor. In Fig. 6, the WRR scheduling policy shows an additional 38.1 percent energy saving (on an average) compared to the FCFS scheduling policy.

In Fig. 7, the variation in energy savings in different modes of operation is plotted for two different test cases. Both Synthetic III and integrated multimedia test cases are considered here as they consist of three modes each and Table 5 shows their mode configurations. Significant energy saving (up to 20 percent) is observed at different modes of operation. A reduction in service rate from 1 to 0.8 shows further increase in energy savings at different modes. Such studies are important for battery-operated systems when a set of functionality can be withdrawn to enhance the battery life.

Fig. 8a demonstrates the service rate variations at a node during different intervals for a given monitoring interval ($z = 12$). It is seen that for different scaling factors the service rate is degrading gracefully which results in reducing the energy savings. This indicates that it is possible to apply maximum power constraints in the system to limit the power consumption using the proposed technique. In Fig. 8b, we present the relationship between the monitoring interval $z$ and the service rate. It may be observed that there is a “sweet spot” on the monitoring interval value that corresponds to an optimal service rate for obtaining the maximum energy savings. For small values of monitoring interval, the service rate is high since we do not get enough slack due to runtime variations. But, for large value of $z$, the frequency of monitoring increases. The overhead of frequent monitoring outweighs the benefits from runtime variations. Such a trend in the service rate variation is noticeable for different values of power constraints as shown in Fig. 8b.

We considered the XScale processor as one of the processing nodes to determine the periodic service rate (PSR) overhead at a node. A cosimulation framework was established with the XScale processor as a node which communicates with our event driven simulator that simulates the entire distributed system consists of ten nodes connected in linear fashion. As the number of task sets increase in the system, the PSR overhead increases as depicted in Fig. 9. For 30 task sets in the Xscale node, the PSR overhead was found to be 260 microseconds. The PSR calculation is done concurrently during the task processing and the result of which is used for setting the voltage/frequency during the next interval. Overhead of voltage/frequency setting is of the order of 150 microseconds for XScale embedded processor and considered to be negligible compared to the interval period $I$ considered in the experiments.

### 4.3 Case Studies

In this section, we present a simple case study on a Star Navigation Controller that can operate in multiple modes. The controller has its inputs from Gyro, CCD cameras, and a telemetry unit as is shown in Fig. 10. The implementation of the controller is done in a distributed framework and each module in the system constitutes a processing node. The Star Identification application involves computation at the modules $S_1$, $S_2$, $S_3$, $S_7$, and $S_8$. This computation is based on stars’ orthogonal views from the two cameras during a lost in space condition and has a computational

<table>
<thead>
<tr>
<th>Test Cases</th>
<th>Number of Nodes</th>
<th>Number of Task sets</th>
<th>Number of Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthetic I</td>
<td>3</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Synthetic II</td>
<td>5</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>Synthetic III</td>
<td>10</td>
<td>30</td>
<td>3</td>
</tr>
<tr>
<td>Integrated multimedia</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>DSP</td>
<td>16</td>
<td>31</td>
<td>1</td>
</tr>
</tbody>
</table>

Fig. 5. The relationship between system task set drop and normalized service rate.
The Recursive Star Identification is based on a fast prediction that uses star images involving computation on the modules \{S1, S5, S3, S7, and S8\} and has a computational deadline of 0.1 seconds. The above two applications are mutually exclusive. At any point of time, only one of these two applications is active. The Positional Correction application, based on the gyro data processing, involves computation on the modules \{S4, S3, S7, and S8\} and has a deadline of 0.01 seconds. The Telemetry Interface application involves computation on the modules \{S6 and S8\} and has a computational deadline of 0.001 seconds. At any point, the system operation is characterized by the set of applications executing at that instant.

In order to guarantee the processing deadlines of an application (or its functions), the task sets are established by mapping the functional modules onto appropriate
processing nodes. Table 6 shows the various task sets and their parameters at different nodes. We allow heterogeneous scheduling strategies (FCFS and WRR here) in the system to examine possible energy savings at various nodes. The choice of using a scheduling policy at a node is arbitrary in this study.

The actual energy drawn out of the battery is determined using $\int_{0}^{B} p(t)dt$ where $B$ is the length of any arbitrary busy interval and $p(t)$ is the instantaneous power. Table 7 depicts the module-wise energy savings compared to the ones without any slack management strategy. In order to demonstrate the effects of varying task set, at node S8, we examined the case with and without the $\prod_4$ task set corresponding to the presence/absence of telemetry signals. The last row in Table 6 shows the effect of energy savings. A significant 18 percent energy savings is obtained when the $\prod_4$ task set is not present at node S8. One can extend such studies across the system and evaluate the variation on the energy profile with dynamically changing the workload in a distributed system. Further, different scheduling combinations across the nodes will result in different energy savings and can be investigated.

![Fig. 9. Periodic service rate overhead with number of task sets.](image)

![Fig. 10. StarNav controller architecture.](image)

**TABLE 6**

<table>
<thead>
<tr>
<th>Task set</th>
<th>Modules Involved</th>
<th>Task set Parameters (msec)</th>
<th><em>(P1); (C1); (D1)</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Starld $\prod_1$</td>
<td>{S1, S2, S3, S7, and S8}</td>
<td>{1000, e, e, e, e}; (80, 400, 0.2, 0.4, 0.2); (e, e, e, e, 1000)</td>
<td></td>
</tr>
<tr>
<td>ReStld $\prod_2$</td>
<td>{S1, S5, S3, S7, and S8}</td>
<td>{100, e, e, e, e}; (80, 0.4, 0.2, 0.4, 0.2); (e, e, e, e, 100)</td>
<td></td>
</tr>
<tr>
<td>PostCor $\prod_3$</td>
<td>{S4, S3, S7, and S8}</td>
<td>{10, e, e, e}; (0.25, 0.2, 0.4, 0.2); (e, e, e, 10)</td>
<td></td>
</tr>
<tr>
<td>TeleInt $\prod_4$</td>
<td>{S6 and S8}</td>
<td>{1, e}; (0.3, 0.2); (e, 1)</td>
<td></td>
</tr>
</tbody>
</table>
TABLE 7
Module-Wise Energy Savings (* without $\Pi_2$, + with $\Pi_2$)

<table>
<thead>
<tr>
<th>Module</th>
<th>Scheduling Policy</th>
<th>Worst-Case delay</th>
<th>% Energy Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>FCFS</td>
<td>80 msec</td>
<td>49.35%</td>
</tr>
<tr>
<td>S2</td>
<td>FCFS</td>
<td>400 msec</td>
<td>67%</td>
</tr>
<tr>
<td>S3</td>
<td>WRR (0.3, 0.7)</td>
<td>(0.4, 0.32) msec</td>
<td>56.3%</td>
</tr>
<tr>
<td>S4</td>
<td>FCFS</td>
<td>0.25 msec</td>
<td>96.38%</td>
</tr>
<tr>
<td>S5</td>
<td>WRR</td>
<td>0.4 msec</td>
<td>31.6%</td>
</tr>
<tr>
<td>S6</td>
<td>WRR</td>
<td>0.3 msec</td>
<td>51.05%</td>
</tr>
<tr>
<td>S7</td>
<td>FCFS</td>
<td>0.8 msec</td>
<td>31.7%</td>
</tr>
<tr>
<td>S8</td>
<td>FCFS</td>
<td>0.4 msec*</td>
<td>0.6 msec+</td>
</tr>
</tbody>
</table>

5 CONCLUSION
In this paper, we presented a low-power slack distribution technique that addresses the issue of guaranteeing the end-to-end and local deadlines of the messages in a multimode real-time distributed embedded systems. The proposed analytical technique uses a comprehensive traffic description function that provides adequate information about the worst-case traffic behavior at various nodes in the network. It allows runtime task variations in real-time distributed embedded systems and reduces overall energy consumption. To demonstrate the effectiveness of our approach, we developed an event driven simulator, tools, and a cosimulation framework. Both synthetic benchmarks and real world test cases were used in the experiments. The proposed approach periodically analyzes the traffic pattern and determines the best possible service rate to take advantage of the runtime variations in execution times, idle intervals, and the nonarrival of tasks to save the energy. The experimental results indicate that significant energy savings are obtained when the proposed technique is used for real-time multimode embedded systems. It may be noted that task sets considered in this work are canonical, without any branches or forks. For future work, we shall consider the task sets of noncanonical type, the impact of task criticality while admitting the task set, and context switch overhead due to the mode changes.

ACKNOWLEDGMENTS
The authors sincerely acknowledge the contributions of Rajesh Prathipati in preparing the initial draft and assistances from Subrata Acharya for verifying the experiments and editing the manuscript during its revision. The authors also duly acknowledge the assistance from Nitesh Goyal for his help during cosimulation. This work was partially supported by a research contract from NASA to the first author.

REFERENCES
Rabi N. Mahapatra received the PhD degree in computer engineering from the Indian Institute of Technology at Kharagpur, India, in 1992. He was an assistant professor in the E&ECE Department at IIT Kharagpur until 1995. Currently, he is an associate professor with Department of Computer Science at Texas A&M University, College Station, Texas. He has directed the Hardware Software Codesign Research Group at Texas A&M University since 2001. His current research interests include embedded system codesign, system-on-chip, VLSI design, and computer architectures. He has published more than 55 papers in reputed journals and conference proceedings. His recent publications can be found at http://faculty.cs.tamu.edu/rabi/. He is a senior member of the IEEE.

Wei Zhao received the MSc and PhD degrees in computer and information sciences at the University of Massachusetts at Amherst in 1983 and 1986, respectively. He is an associate vice president for research and a professor of computer science in Texas A&M University. He is the director for the Division of Computer and Network Systems in the US National Science Foundation. He completed his undergraduate program in physics at Shaanxi Normal University, Xian, China, in 1977. During his career, he has been a faculty member at Amherst College, the University of Adelaide, and Texas A&M University. As an IEEE fellow, Wei Zhao has made significant contributions in distributed computing, real-time systems, computer networks, and cyber space security. His research group has been recognized by receiving various awards and prizes, including the outstanding paper award from the IEEE International Conference on Distributed Computing Systems, the best paper award from the IEEE National Aerospace and Electronics Conference, an award on technology transfer from the Defense Advanced Research Program Agency, and the second prize in the international ACM student research contest. He is an inventor for two US patents and has published more than 220 papers in journals, conferences, and book chapters. He is the founding director of the Texas A&M Center of Information Security and Assurance, which has been recognized as the Center of Academic Excellence in Information Assurance Education by the National Security Agency. He has also been active in professional services. He has served on editorial board of technical journals, including the IEEE Transactions on Computers and the IEEE Transactions on Parallel and Distributed Systems. He is the chair-elect for the IEEE Technical Committee of Real-Time Systems. He has chaired more than 10 international conferences including the IEEE Real-Time Technology and Applications Sympoisum, the IEEE Real-Time Systems Symposium, and the IEEE International Conference on Distributed Computing Systems.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.