Energy-Aware Soft Real-Time Scheduling for Multi-Radio Embedded Devices *

John M. Calandrino[1], Nissanka B. Priyantha[2], Jie Liu[2], and Feng Zhao[2]
[1]Department of Computer Science, The University of North Carolina at Chapel Hill, Chapel Hill, NC
[2]Microsoft Research, One Microsoft Way, Redmond, WA

Abstract

This paper presents an energy-efficient scheduling algorithm for the data-communications of soft real-time periodic tasks on multi-radio embedded devices. To cope with the dynamic fluctuation of channel conditions, a feedback mechanism that monitors radio throughput is introduced to guarantee real-time behaviors. The paper also provides a formal analysis on the scheduling, exploring the relationship between tardiness bounds and energy savings and that between network stability and (m, k)*-firm deadline guarantees. This approach is applicable to real-time communication scheduling problems in which background interference and other environmental factors are not known* a priori. *The algorithm is evaluated in the QualNet simulator, using data measured from radio hardware.*

1 Introduction

In this paper, we present an algorithm for the energy-efficient scheduling of periodic soft real-time tasks on multiple, heterogeneous radios located within a single node of a network. The algorithm allocates tasks to radios such that each block of application data is transmitted by a radio within given timing constraints. Such constraints will take a form similar to those that specify the processing requirements of periodic tasks, *e.g.*, "send twenty kilobytes of data every three seconds." Furthermore, this mechanism is implemented entirely within a single node, rather than within a group of nodes collaborating to meet their data-sending requirements. To react to changes in the effective radio throughput due to various environmental factors, we use a feedback loop with a dynamic radio throughput estimator. This feedback-based scheduler enables us to make efficient scheduling decisions with little *a priori* knowledge about the networking environment.

Some emerging applications for mobile and sensing devices call for long-lived data transfers between a device and the wireless networking infrastructure. For example, many music and video players wirelessly stream multimedia content from content servers. PDAs with wireless capability run applications that periodically receive updates on stock prices, traffic, and news stories. Sensors monitoring various physical phenomena periodically send sensed data for analysis by scientists. We note that, even for applications that stream data, the actual data transfer is composed of a "periodic" stream of packets. Although the real-time nature of these applications require that the data be delivered within some deadline, these deadlines are soft in that a certain amount of deadline tardiness may be tolerated, as long as it is bounded. For example, the internal buffering of received data enables a video playback application to tolerate a bounded variation in communication delay. We also note that the time scale of concern for sending data is substantially different than those commonly considered for processor scheduling—we consider periods and other time factors in units of seconds instead of milliseconds or microseconds.

Long-running wireless data transfers result in significant energy consumption by radios, thus substantially affecting the battery life of mobile devices and sensor nodes. One major opportunity for minimizing energy consumption is related to the availability of an increasing number of different radio technologies within mobile devices, with varying energy and bandwidth characteristics. For example, a typical smartphone may contain up to 4 different types of radios for ubiquitous connectivity [9]; similarly, emerging sensor network platforms also support multiple radios [18, 17]. Given a device with such multiple radio technologies, and a collection of real-time tasks with different data rates and real-time constraints, we present a scheduling algorithm that assigns those tasks to different radios such that we minimize the radio energy consumption while maintaining soft real-time guarantees for task data transmission.

In the rest of this paper we consider a device with two radios, one 802.11 radio and one 802.15.4 radio. Based on typical device characteristics, the power consumption of the 802.11 radio is an order of magnitude higher than the 802.15.4 radio when both radios are on and idle; however, the additional energy beyond the idle energy that is required to send a unit of data is higher for an 802.15.4 radio than

*This research was conducted while the first author was at Microsoft Research for a summer internship.

for an 802.11 radio, since the 802.11 radio sends data much faster resulting in a much smaller transmission time. This combined with the larger start up energy of 802.11 radio makes the 802.15.4 radio more energy efficient for smaller payloads, while the 802.11 radio is more energy efficient for larger payloads. Since the 802.11 radio consumes a lot of energy when powered on, we want to keep it powered down when not sending data—if we need to keep a radio on and listening (for requests from remote radios for example), then the 802.15.4 radio should be used for this.

Given these radio characteristics, supporting real-time requirements would seem straightforward—turn on the faster 802.11 radio immediately before sending data, and off immediately after. In fact, supporting real-time constraints seems completely decoupled from the problem of maximizing energy efficiency. Unfortunately, this is not the case. The 802.11 radio requires a non-negligible amount of time to transition from the off-to-on state. Therefore, when data needs to be sent out on a schedule to support some real-time tasks, it may not be possible to turn off the radio during every idle interval. Furthermore, the 802.11 radio consumes a considerable amount of energy when *transitioning* from the off-to-on state, so it must be determined whether it is *energy efficient* to power down the radio, even when the idle interval is large enough.

Since short idle times between data transmissions may prevent us from turning off the radio, a possible approach to reducing energy consumption is to buffer a large amount of data, power on the 802.11 radio, and send out the data in the buffer. This approach saves energy by eliminating short idle times and multiple radio power on events associated with sending small amounts of data. This approach also allows us to exceed the threshold payload size that makes the 802.11 radio more efficient than the 802.15.4 radio. However, we cannot blindly use this approach since we may violate application timing guarantees. Instead, we apply this idea to real-time task scheduling in a way that still allows formal guarantees to be made, by showing a direct relationship between deadline *tardiness bounds* and *energy savings*. We show that if we can tolerate a small amount of deadline tardiness, then in most cases, we can substantially improve the energy efficiency of the radios. We also show that, as we increase deadline tardiness, we observe diminishing returns in energy savings; however, we continue to observe benefits due to a reduction in the number of times we turn on the radio to send data over any time interval.

Another concern when sending data using radios is that the *throughput*, or the effective data rate a radio can support, fluctuates substantially over time. This contrasts with processor scheduling, where the capacity of the processor to do work in a given power mode is typically known *a priori*. As a result, some feedback mechanism is required to estimate the current throughput of each radio and to determine the real-time workload that we can support at the current time. Without such a feedback mechanism, any scheduling algorithm will be ineffective in practice, especially when used in portable devices where the interference from other devices and environmental factors is very dynamic. Our feedback mechanism detects throughput overestimates by queue overflow events resulting from the mismatch between the current throughput estimate and the rate at which a given radio is sending out data. Since the effects due to throughput underestimates cannot be directly observed, we periodically increase our throughput estimate by a constant amount to avoid persistent throughput underestimates. Note that a decrease in the throughput estimate might result in a system that can no longer support its real-time workload. In this case, we drop tasks until the current workload can be supported given the current throughput estimate.

Naturally, instability in our network or operating environment that causes throughput to change will make it more difficult to provide real-time guarantees. In our analysis, we seek to contain the impact of throughput uncertainty on our real-time workload by representing it as an alternating series of stable and unstable intervals, for which we can make (m, k)-firm guarantees [8] (at least m of every k consecutive jobs will meet their timing requirements). In environments where the throughput uncertainty is too great, it may be difficult for any such guarantees to be made.

Contributions. Our contributions are as follows.

- We present an algorithm for scheduling periodic soft real-time tasks on multiple, heterogeneous radios. Evaluation shows that this algorithm can substantially reduce energy consumption while being aware of the impact that the large wake-up time of a high-power radio has on the real-time guarantees that can be provided.

- We show that if we "aggregate" work over some number of hyperperiods into a single busy interval, then we can take advantage of the resulting larger idle interval to further improve energy efficiency, in exchange for an increase in (bounded) deadline tardiness. This result demonstrates a tradeoff between energy efficiency and deadline tardiness that we explore both analytically and empirically.

- We thoroughly evaluate our feedback mechanism, which handles changes in perceived throughput, in terms of its ability to minimize the disruption to our real-time workload due to network interference. We make several claims about how feedback parameters relate to how well we support our real-time workload.

- We state a relationship between network stability and (m, k)-firm guarantees. We measure the sizes of stable and unstable intervals in experiments, and use them to predict what types of (m, k)-firm guarantees we can make in practice.

We implemented and evaluated our algorithm in QualNet, a network simulator that simulates the entire wireless network stack. We also used it to simulate multiple patterns of background traffic, including both constant and fluctuating levels of interference over the course of an experiment.

The rest of this paper is organized as follows. Sec. 2 discusses related work. Sec. 3 introduces our task and radio models. Sec. 4 describes our algorithm. Sec. 5 presents two analytical relationships—the first between energy efficiency and deadline tardiness, and the second between network stability and (m, k)-firm deadline guarantees. Sec. 6 presents an experimental evaluation of our method in QualNet. Finally, Sec. 7 concludes and discusses several areas of future work.

2 Related Work

In this section, we discuss related work. Due to the vast amount of prior work that is related to this research, we only attempt to provide an overview of this work with examples.

2.1 Soft Real-Time Guarantees

There exist various types of soft real-time guarantees that can be made when it is not necessary for all real-time tasks to always meet their deadlines. Two types of soft real-time guarantees are used in this paper: *tardiness bounds* and (m, k)-*firm guarantees*. A tardiness bound indicates the maximum amount by which the deadline of any job may be missed. A recent paper has shown that, for a large class of multiprocessor scheduling algorithms including most deadline-based approaches, tardiness is always bounded [13]. An (m, k)-firm guarantee [8] states that for a sequence of k consecutive jobs, at least m of those jobs will meet their timing requirements, thus producing a useful result. Such a guarantee also implies that $k - m$ jobs in any sequence of k consecutive jobs may miss their deadlines without causing the system to fail. In this paper, we make use of this type of guarantee when considering the impact of network instability on real-time guarantees.

2.2 Feedback Control in Real-Time Systems

Work also exists in the area of incorporating feedback control loops into real-time systems. For example, work in [16, 24] adjusts the utilizations of periodic real-time tasks within some range in response to fluctuating demands in a (distributed) system. Additionally, the work in [14, 15] uses feedback control theory to determine the execution costs of periodic tasks given an initial estimate of such costs. Both works were some of the first to incorporate feedback control into the periodic task model. In this work, we take a similar approach to providing feedback control within our system, with two major differences. First, in our system, while we can adjust to a dynamic workload through the use of an admission control protocol, we are primarily attempting to estimate "supply" in terms of achieved throughput by a particular radio, instead of demand. Therefore, the initial estimates in our system are for the throughput of each radio rather than the execution requirements of each admitted task. We also respond directly to dropped packets and output queue overflow rather than waiting until the end of a sampling period to react. Second, our method places no bounds on the utilizations of tasks as a result of fluctuating throughput—however, tasks may need to be "dropped" from a radio that is over-utilized as a result of a change in throughput. Such "dropped" tasks can be re-admitted onto another radio, if possible, via our admission control protocol.

2.3 Real-Time Guarantees in Networks

There exists much previous work related to providing real-time or QoS guarantees in networks, particularly sensor networks—examples of recent work include [20, 6, 4, 12, 11, 5]. The most important difference between the majority of this work and ours is that considerable information about the state of the entire network, such as network topology, link latencies, and link capacities, is usually required to make scheduling decisions for a collection of nodes in the network. By contrast, our mechanism is per-node, where each node relies only on *local* information to adapt to perceived changes in throughput as a result of fluctuating network loads or environmental factors. As a result, it is better suited to environments where either much less is known about the network *a priori*, or it is very difficult to get such network-wide state information. Additionally, much prior work is not directly concerned with supporting the periodic task model for scheduling the data-sending requirements of real-time tasks; instead, the focus is on bounding or reducing network latency, improving reliability, increasing network capacity, or otherwise *facilitating* real-time support.

2.4 Energy-Aware Dual-Radio Nodes

Finally, some previous work has examined the use of multiple radios to reduce energy consumption [22, 23, 19]. In this approach, a high-power, 802.11 radio is powered-down when not in use, and a low-bandwidth, low-power secondary radio listens for incoming messages and "wakes" the 802.11 radio as necessary. Our work is different since we are interested in scheduling tasks on *all* available radios.

In the real-time community, an energy management framework has been proposed for periodic tasks that takes a system-level approach by including the impact of components other than the CPU [3]. The energy consumption of the system is divided into frequency-dependent and frequency-independent components. Our work helps such a model to achieve better results for multi-radio nodes by reducing the energy consumed by the radios.

3 Models

In this section, we introduce our task and radio models, including several assumptions made throughout this paper.

3.1 Task Model

We consider the scheduling of a system τ of *periodic tasks*, denoted $T_1, \ldots, T_N$, on two radios (though this could be generally applied to m radios). T_i is specified by its *worst-case* (*per-job*) *execution cost*, e_i, and its period p_i. The j^{th} job (or invocation) of task T_i is denoted T_i^j. Such a job T_i^j becomes available at its release time, or the start of the j^{th} period. A job should complete execution by its absolute deadline, or its release time plus its period. Otherwise, it is *tardy*. Task T_i's *utilization* is given by e_i/p_i. The *hyperperiod* of τ is the least-common-multiple (LCM) of all task periods. For a synchronous task set (all tasks start at the same time), we often consider the scheduling of τ over a single hyperperiod, since no job of any task can be released in one hyperperiod and have its deadline in the next. Instead, all tasks will have a job deadline and release at each hyperperiod boundary.

We make the following assumptions about the tasks in τ. First, tasks can withstand occasional tardiness or dropped jobs, thus it is reasonable to consider soft real-time guarantees. Second, e_i is a function of the amount of data that must be sent k_i, and the current throughput estimate of the radio to which T_i is assigned—k_i is known *a priori*, but e_i is not. Tasks may have execution costs on the 802.15.4 radio that result in task utilizations exceeding one—such tasks must instead be assigned to the 802.11 radio. Third, the specified requirements say nothing about the *CPU* resource needs of a task, only its radio needs. CPU scheduling can either be considered separately (as long as a buffer always contains sufficient data to send), or our methods can be incorporated within a system-level scheduling framework. Finally, if data packets associated with a job are lost during transmission, we formally consider that job "dropped" (though less conservative assumptions could work, too); however, we still attempt to send the remaining data associated with that job, hopefully to minimize the disruption within the system as a result of losing data. We do *not* attempt to re-send packets, as doing so might result in a task over-run and negatively impact jobs that are otherwise executing correctly. This upholds the paradigm that failures related to one task should not affect the other tasks in the system.

3.2 Radio Model

We assume dual-radio nodes containing both 802.11 and 802.15.4 radios, though our methods can be generalized to more than two radios. Energy and time measurements for the radios are shown in Table 1. Note that the measurements here reinforce the statements made in Sec. 1. The energy to send a packet is the *total energy* consumed by the radio for the interval of time that it is sending the packet—the on power stated is only applicable when the radio is idle. Nearly all values represent measurements on real radios (a CC2420-based 802.15.4 radio and a PRISM 2.5 chipset-based 802.11 radio) in the mPlatform sensor node [17] (these measurements will be discussed in greater detail in a future paper). The exceptions are off-to-on energy and time for the 802.11 radio, which are estimated with help from values provided in [21]. We use these values when performing energy accounting in our experiments—we periodically update energy consumption depending on the radio state, and also perform updates in response to sending packets or turning on the radio.

	802.11 Radio	**802.15.4 Radio**
Radio on power	742.5 mW	1.65 mW
Radio off power	1 μW	1 μW
Max. payload size	1500 bytes	117 bytes
Energy to send packet (payload of X bytes)	$(0.617 \cdot X + 168.3)$ μW	$(2.6 \cdot X + 50)$ μW
Time to send packet (payload of X bytes)	$(0.733 \cdot X + 200)$ μs	$(41 \cdot X + 910)$ μs
Off-to-on time	1 s	negligible
Off-to-on energy	690 mW	4 μW

Table 1: Radio energy- and time-related attributes.

4 Scheduling Algorithm

We now present our algorithm, a diagram of which is shown in Fig. 1, by discussing each component individually.

4.1 Task Scheduling

We use the partitioned earliest-deadline-first (EDF) algorithm to schedule task sets. While a partitioned approach can result in system under-utilization due to bin-packing limitations (especially on heterogeneous platforms such as uniform multiprocessors—see [7]), it greatly simplifies the scheduling of tasks on heterogeneous radios, by allowing each radio to be considered separately when handling network instability and the resulting changes in estimated throughput. The EDF algorithm allows us to fully utilize each radio, effectively maximizing the size of each "bin." EDF is also work-conserving, which means that a radio is never idle when there is data ready to be sent. We exploit this property of the algorithm in Sec. 5 when determining the relationship between tardiness bounds and energy consumption.

Admission control. Our admission control policy seeks to minimize the additional energy that will be consumed as the result of admitting a real-time task. We first determine the set of radios that could admit the task without being over-utilized. (Recall that task execution times could vary substantially for each radio.) If this set is empty, then we must reject the task. Otherwise, we estimate the additional energy that would be required to support this task on each radio in the set over a large interval. The interval that we choose is equal to the hyperperiod of the tasks that are currently admitted onto *any* radio—this allows the comparison of energy values for different radios to be meaningful. We include in this estimate the additional energy required to send task data, and when possible, the impact of smaller idle intervals on our ability to conserve energy by turning off radios. (Note that

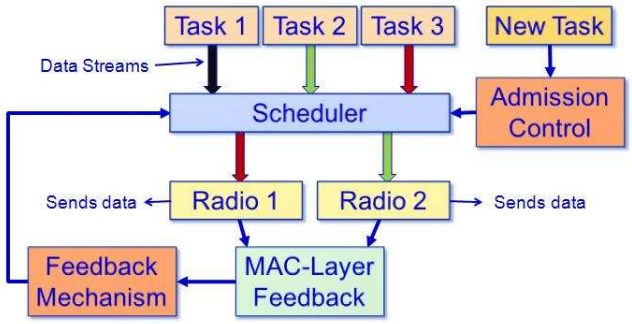


Figure 1: The components of our algorithm (three admitted tasks).

we could get a better estimate by *simulating* the scheduling of tasks over the hyperperiod; however, this would be impractical in many cases.) We then admit the task onto the radio for which the energy estimate is lowest. Note that it may sometimes be useful to restrict the times at which a new task can be admitted onto a radio, in order to provide stronger analytical guarantees. In particular, we might want to admit tasks only at times that result in job release times and deadlines that are equivalent to those in a synchronous task system, which allows the tardiness result in Sec. 5 to hold.

Turning off radios. During each scheduling decision, radios that are idle may be turned off, provided that it is both safe and efficient to do so. It is *safe* to turn off a radio if the off-to-on time of the radio is smaller than the interval from now until the next job release of any task. It is *efficient* to turn off a radio if the energy consumed over the idle interval when turning the radio off and back on (a function of the power consumed when off and the off-to-on energy) is less than the energy consumed if the radio is left on over the same interval. Note that this decision is made online, so in the case that jobs often complete much sooner than their worst-case execution times would predict, we may be able to save considerably more energy by turning off a radio than would be predicted through analysis.

4.2 Feedback Mechanism

The throughput of a radio can fluctuate unpredictably over time, and this fluctuation directly impacts the real-time guarantees that we can make. Thus, a feedback mechanism is necessary for meaningful real-time guarantees to be established. At the heart of our mechanism is a throughput estimator with behavior similar to the *additive-increase-multiplicative-decrease* (AIMD) protocol used to estimate the capacity of a TCP connection [10].

Direct response mechanism. The direct response mechanism is triggered by output queue overflow events. These events occur when a packet must be dropped since it cannot be placed in the output queue, due to insufficient space. Such events indicate a mismatch between the throughput estimate of our real-time scheduler and the actual rate at which the radio is able to send data. Thus, we immediately decrease our throughput estimate by a multiplicative factor in response to such events, which should reduce stress on the radio and allow the system to re-stabilize. Immediately after responding to an overflow event, we ignore further such events for a short interval in order to give the system some time to stabilize, after which we again respond to such events by adjusting our throughput estimate.

Dropping tasks. As throughput decreases, the execution costs of tasks assigned to the radio will increase. As a result, the radio might become over-utilized. If this occurs, then tasks must be dropped. When faced with this situation, we drop tasks in the following order.

- Drop all tasks with utilizations greater than one (they cannot be supported).
- Drop tasks in increasing order of some pre-determined *drop priority*, so that less important (*i.e.*, lower drop priority) tasks are dropped first.
- In the case of a tie in drop priority, or when drop priorities are unspecified, we drop the task with utilization closest to the current over-utilization of the radio.

The final rule should drop the smallest number of tasks when the radio is heavily over-utilized, and typically ensures that the radio does not become heavily under-utilized as a result of dropping the wrong tasks. For example, two tasks of utilization 90% and 20% will over-utilize the radio by 10%; our approach would recognize that it makes more sense to drop the 20%-utilization task and achieve nearly-full radio utilization. Note that if we drop a task that is currently executing, then we need to make a scheduling decision. Additionally, we attempt to re-admit dropped tasks whenever the throughput estimate of a radio increases or the size of the real-time workload decreases.

Periodic mechanism. The periodic mechanism increases the throughput estimate by some constant amount at every quantum boundary, when no queue overflow events were observed in the previous quantum, and the previous quantum was fully utilized (since only then can be we reasonably sure that data is being successfully sent by the radio at the estimated rate). This increase occurs before scheduling or admission control decisions are made, to allow tasks to be admitted during this quantum as a result of the increase. Note that as throughput increases, the execution costs of tasks assigned to the radio can only decrease, so there is no need to drop tasks in this case.

5 Formal Analysis

In this section, we present two theorems that formalize some of the relationships in our system. The first theorem demonstrates a tradeoff between energy consumption and deadline

tardiness, and the second theorem establishes (m, k)-firm guarantees in the presence of network instability.

5.1 Energy vs. Tardiness Tradeoff

We first discuss a relationship between deadline tardiness and energy consumption. Given the power characteristics of certain radios, particularly 802.11 radios, a large idle interval may be required in order to making turning off the radio energy efficient. As a result, there may exist task sets for which it is impossible to turn off the radio to save energy, as a large enough idle interval cannot exist if all deadlines are to be met (even if a work-conserving scheduling algorithm is used).

If deadlines may be missed by bounded amounts, then this gives us additional scheduling flexibility that can be exploited to save energy. For a given scheduling hyperperiod, we can combine all of the idle time into a single idle interval of length I, and force the radio to be idle for I time units at the beginning of the hyperperiod. Since the remaining time in the hyperperiod is still sufficient to complete all jobs by the end of the hyperperiod, such intervals are still self-contained in that we only need to consider the schedule one hyperperiod at a time. Furthermore, no task will miss its deadline by more than I time units. We can further extend this method to combine idle time over multiple hyperperiods, as necessary to get the desired idle interval. Doing so will increase deadline tardiness, but only by the size of the idle interval generated. For example, if we combine idle time over ten hyperperiods into a single three-second interval, then jobs may miss deadlines by at most three seconds. Using this method, we can generate large enough idle intervals so that it becomes energy-efficient to turn off the radio, at the cost of a higher tardiness bound. We refer to this method as *idle-time aggregation* over X hyperperiods, where $X > 0$.

We now state and prove the following theorem, which formally states the relationship between idle-time aggregation, deadline tardiness, and energy savings. Let I be the aggregated idle time over a single hyperperiod of a schedule, W_e and W_t be the off-to-on energy and time of the radio, and $\mathcal{I}$ and $\mathcal{O}$ be the power consumption of the radio when idle and off, respectively.

Theorem 1 *If we perform idle-time aggregation over X hyperperiods, then* **(a)** *deadline tardiness is at most $X \cdot I$; and* **(b)** *the energy savings $\mathcal{E}$ over some Z-hyperperiod interval, where X divides Z, is as follows.*

$$\mathcal{E} = \begin{cases} 0, & \text{if } X \cdot I \leq \frac{W_e - W_t \cdot \mathcal{O}}{\mathcal{I} - \mathcal{O}} \\ Z \cdot I \cdot (\mathcal{I} - \mathcal{O}) - & \\ \quad (Z/X) \cdot (W_e - W_t \cdot \mathcal{O}), & \text{otherwise.} \end{cases}$$

Proof: For part **(a)**, consider a synchronous, EDF-schedulable task system τ. Hyperperiod boundaries are clearly defined for such a task system, since demand from one hyperperiod does not exist in the next. To create a schedule in which idle-time aggregation is employed, we start with an EDF schedule over X hyperperiods, shift left all of the idle periods in the schedule, and shift right all of the busy periods in the schedule. (Note that we cannot shift the busy periods left and the idle periods right, unless jobs can execute before their release times.) Since all modifications to the schedule are contained within the X-hyperperiod interval, the amount of idle and busy time within the interval remains the same. Thus, all jobs still complete execution by the end of the X-hyperperiod interval, and we can consider each such interval separately due to this fact.

We next consider the impact of idle-time aggregation. First, consider the case where we have an idle interval of length $X \cdot I$, and all job releases and deadlines have been pushed back by $X \cdot I$ time units. In this case, the entire schedule is simply shifted right by $X \cdot I$ time units, and all jobs meet their new deadlines. Now, assume that all jobs were released at their original times. This is equivalent to early releasing all jobs $X \cdot I$ time units before their shifted release times. It has been shown in [2, 1] that early-releasing jobs in global, deadline-based scheduling methods such as EDF has no negative impact on the ability of jobs to meet their deadlines, thus all jobs still meet their new deadlines. Finally, a job that meets its new deadline will only miss its original deadline by at most $X \cdot I$ time units. This, all jobs released at their original times within the X-hyperperiod interval will miss their deadlines by at most $X \cdot I$ time units due to idle-time aggregation. (Note that a similar argument relating shifting job releases and deadlines, early-releasing, and tardiness bounds exists in [1].)

We prove part **(b)** as follows. Our idle interval length is $X \cdot I$; in order to achieve energy savings by turning off the radio, $\mathcal{I} \cdot X \cdot I > \mathcal{O} \cdot (X \cdot I - W_t) + W_e$. That is, the energy consumed by keeping the radio on and idle for $X \cdot I$ time units must be greater than the energy to turn it off and back on over the same amount of time. Solving for $X \cdot I$, we get $X \cdot I > \frac{W_e - W_t \cdot \mathcal{O}}{\mathcal{I} - \mathcal{O}}$.

If $X \cdot I$ is less than this value, our energy savings will be zero. Otherwise, we will achieve energy savings for a Z-hyperperiod interval equal to the energy saved over the interval by turning the radio off (versus keeping it on and idle) minus the energy spent turning the radio on Z/X times, which is equal to the equation shown in the second part of the definition of $\mathcal{E}$ above. As X increases, we achieve diminishing returns as Z/X approaches one. □

5.2 (m, k)-Firm Guarantees and Instability

We next determine the impact of network instability on the real-time guarantees that we can provide. Without loss of generality, we can represent network stability for a radio over time as an alternating sequence of stable and unstable intervals. Stable intervals contain no queue overflow events that would indicate an inaccurate throughput estimate; unstable intervals may contain any number of such events, resulting in lost data. The *minimum stable interval* C is the smallest stable interval observed, and the *maximum unstable inter-*

val γ is the largest observed unstable interval. Note that we can combine small stable intervals and their adjacent unstable intervals into a single unstable interval, when doing so increases C; however, this may also increase γ. We consider this to be a design decision that we explore further in Sec. 6.

We can state (m, k)-firm guarantees for each task in a task set given the C and γ values of the radios on which it will be scheduled—this is formalized in the theorem below. Note that our analysis assumes that any job that overlaps with any unstable interval will be dropped. In reality, this may be too conservative, and we could re-define an unstable interval to be any interval containing an event that would cause a job overlapping that interval to be dropped. This new definition might result in better network stability and (m, k)-firm guarantees.

Theorem 2 *Given a task T_j, C and γ as defined above, $V = (k \cdot p_j) mod (C+\gamma)$, and no deadline tardiness:* **(a)** *no (m, k)-firm guarantee can be made if $C < p_j$, since at least one unstable interval will overlap with every job; and* **(b)** *if $C \geq p_j$, then at most $d = \left(\lfloor \frac{k \cdot p_j}{C+\gamma} \rfloor\right) \times \left(\lfloor \frac{\gamma}{p_j} \rfloor + 2\right) + max(\mathcal{A}, \mathcal{B})$ jobs of any k consecutive jobs are dropped due to dropped packets, where $\mathcal{A}$ and $\mathcal{B}$ are defined below, thus resulting in an (m, k)-firm guarantee where $m = k - d$.*

$$\mathcal{A} = \begin{cases} 1, & \text{if } V \leq C \\ \lfloor \frac{V-C}{p_j} \rfloor + 3, & \text{otherwise.} \end{cases}$$

$$\mathcal{B} = \begin{cases} -1, & \text{if } V = 0 \\ \lfloor \frac{min(V,\gamma)}{p_j} \rfloor + 1, & \text{otherwise.} \end{cases}$$

Proof: First, note that $\lfloor \frac{k \cdot p_j}{C+\gamma} \rfloor$ is the minimum number of $(C + \gamma)$-size intervals that will fully overlap with an interval of length $k \cdot p_j$ representing k consecutive jobs. Within each fully overlapping interval, at most $\lfloor \frac{\gamma}{p_j} \rfloor + 2$ jobs will fully or partially overlap with an unstable interval and be dropped. Multiplying these terms, we get the first part of d; the second part is determined by considering the $(C + \gamma)$-size intervals that may partially overlap at each end of the k-job interval. Two extremes exist: on the left side of the interval, the overlapping interval either is of size ϵ or γ. The former maximizes disturbance on the right side of the k-job interval, while the latter maximizes disturbance on the left side. We now consider each case separately.

Overlap by ϵ: equation $\mathcal{A}$. In this case, we assume that the stable interval comes first in each interval of length $C+\gamma$, therefore exactly one job is dropped by a very small overlapping unstable interval on the left side. On the right side, the overlap is $(k \cdot p_j) mod (C + \gamma) - \epsilon = V - \epsilon$. if $V \leq C$, no additional dropped jobs occur. Otherwise, $\lfloor \frac{V-C}{p_j} \rfloor + 2$ additional jobs could be dropped.

Overlap by γ: equation $\mathcal{B}$. This is equivalent to the case where the start of the first $(C + \gamma)$-size interval lines up exactly with the start of the first job in the k-job interval, thus there is no partial overlap on the left; and the unstable interval comes first in each interval of length $C + \gamma$. On the right, there exists a partially overlapping interval of size $(k \cdot p_j) mod (C+\gamma) = V$. Therefore, at most $\lfloor \frac{min(V,\gamma)}{p_j} \rfloor + 2$ jobs could be dropped in the partially-overlapping interval on the right. If $V = 0$, then no partial overlap exists and no additional jobs are dropped. Due to the exact alignment of both intervals on the left, we must also *subtract* one overlapping job regardless of the value of V, since the unstable interval on the left can partially overlap with only one, not two, jobs.

Combining these cases, we get the equation for d stated in the theorem. □

Note that we could potentially improve our (m, k)-firm guarantees if we had more information about when each task would be running—we might get such information by generating an EDF schedule and overlapping that schedule with our alternating stable and unstable intervals to determine when jobs would be dropped.

Determining C and γ. In our implementation, we calculate C and γ as part of our feedback mechanism. A queue overflow event marks the end of a stable interval (or the continuation of an unstable interval, if we are incorporating the previous stable interval into a longer unstable interval). The start of the next stable interval is set to be the current time plus the interval between which packets are sent (inversely related to the current throughput estimate). The sizes of the resulting stable and unstable intervals are measured, and C and γ are updated accordingly.

Combining Theorem 1 and Theorem 2. Combining the results of both theorems is possible if Theorem 2 is modified to allow for deadline tardiness. Dropped *jobs* will not cause issues for Theorem 1, as demand will only be reduced by such jobs, resulting in reduced deadline tardiness (dropped and re-admitted *tasks*, on the other hand, could be problematic if the system becomes asynchronous). To account for deadline tardiness in Theorem 2, we first determine by how many jobs we can be "behind" as a function of the tardiness bound and the task period, and account for those jobs with a series of intervals of length e_j rather than p_j (for a task T_j). The shorter intervals represent the fact that a series of tardy jobs may execute sequentially without interruption—their execution is not "spaced out" according to task period, since all jobs are beyond their release times. The result should be a somewhat more pessimistic (m, k)-firm guarantee; however, if tardiness bounds are relatively low, or task utilizations are high, the impact may not be dramatic.

6 Experimental Evaluation

In this section, we evaluate our algorithm in terms of its ability to support real-time workloads, save energy, and handle various levels of network interference. All experiments were

conducted within the QualNet network simulator. QualNet simulates the full network stack and exhibits network behavior that is similar to a real network. Combined with background traffic, this results in a highly dynamic networking environment. The QualNet implementation of our algorithm consists of a separate network stack for each radio, with a customized CBR application, for which the (uniform) data rate is a function of the current throughput estimate, at the top of each stack. UDP and IP are used at the transport and network layers; the MAC and physical layers were different for each radio. The energy accounting provided in QualNet was not adequate for our purposes, so we implemented our own accounting using the energy numbers from Table 1.

Our task sets are relatively static—all tasks arrive at the start of the simulation and remain throughout unless they are dropped or re-admitted due to a change in the throughput estimate of a radio. Tasks are periodic and attempt to send the same amount of data every period, *i.e.*, we do not simulate task overruns or underruns; however, these may still occur due to an inaccurate time estimate for sending data, as a result of fluctuating radio throughput. Note that, even though task sets are technically static, network interference causes them to appear to be highly dynamic; thus, we must be able to handle such dynamic behavior.

6.1 Investigating Feedback Parameters

We first investigate how the choice of feedback parameters impacts both energy savings and real-time guarantees. There are three feedback parameters, as follows.

- Throughput increase factor: the amount by which we increase throughput during stable intervals.
- Throughput decrease factor: the multiplicative factor by which we decrease throughput during queue overflow.
- Minimum stable interval: the minimum interval that we record as *stable*, instead of including it as part of an *unstable* interval. These interval sizes do not change scheduler behavior, but impact (m, k)-firm guarantees.

We ran experiments for randomly-generated task sets with a variety of experimental parameters, as follows.

- Interference level: 1 through 5.
  - 1: No background network traffic.
  - 2: Medium—3 additional sender-receiver pairs.
  - 3: High—5 additional sender-receiver pairs.
  - 4: Spike—Medium interference plus a spike.
  - 5: Intense spike—Spike with heavier interference.
- Task periods: between 5-20 sec., with certain periods removed so that the maximum hyperperiod is 720 sec.
- Maximum task utilization: 300% of the initial throughput estimate of the 15.4 radio (3KB/sec).
- System utilization: $\{25, 75\}\%$ the combined initial throughput of both radios ($3 + 30 = 33$KB/sec).
- Throughput increase factor: $\{100, 500\}$ and $\{1000, 5000\}$ bytes/sec. for the 802.15.4 and 802.11 radios.
- Throughput decrease factor: $\{2, 4\}$.
- Minimum stable interval: $\{1, 10, 100\}$ sec.

A sender-receiver pair includes a sender that sends data using an 802.11 radio and an 802.15.4 radio, and a receiver that passively accepts data from those radios. Each 802.11 (802.15.4) radio sends a 1000-byte (100-byte) packet every 20 ms. The size and frequency of interference was chosen based on disruption levels observed from a variety of packet sizes and sending frequencies within a separate set of experiments on real hardware (the mPlatform). Spikes create a 60-second interval of high interference (5 additional sender-receiver pairs) in the middle of each simulation. For an intense spike, the sending frequency doubles during the spike.

For each combination of parameters, we ran ten 300-second experiments. The size of a scheduling quantum was one second, which may seem large, but we allow scheduling decisions to occur between quantum boundaries when a job completes early. Additionally, a scheduling quantum of less than one second would make it difficult to provide the real-time guarantees of interest to us within the simulated network—very little data can be sent by an 802.15.4 radio in a few tens of milliseconds, and we wanted to use the same quantum size for both radios. An exploration of smaller quantum sizes, especially for the 802.11 radio, is left as future work.

Results. We first discuss results related to task performance. First, no deadline misses were observed for jobs that were not dropped. Second, as we can see from Fig. 2, aggressive throughput increase and decrease factors generally resulted in worse task performance than conservative factors, as might be expected. (Aggressive (conservative) throughput factors are 500 bytes/sec. (100 bytes/sec.) increases and 4-fold (2-fold) decreases.) Tasks are dropped more frequently (inset **(a)**), and maintaining a sending rate for a task is more difficult (inset **(b)**), when aggressive parameters are used, especially when the system is 75% utilized (even more so when interference is heavy). The exception is for the case with no background traffic, where aggressive parameters slightly outperform conservative ones. Since simulations stop at a somewhat arbitrary time (300 seconds), we expect that the last job of most tasks will not execute completely, thus 100% of data will not be sent; however, values very close to 100% should be possible, and with higher throughput estimates, are more probable. Third, in almost all cases, aggressive factors result in more dropped jobs (inset **(c)**), especially in the cases where an interference *spike* is observed, and more aggressive factors could result in an "overreaction" to a temporary loss of throughput. In cases where dropped jobs actually decrease when we use the aggressive factors, we speculate that the decrease on system load caused by the dropping of *tasks* results in a less-utilized system where *jobs* are dropped

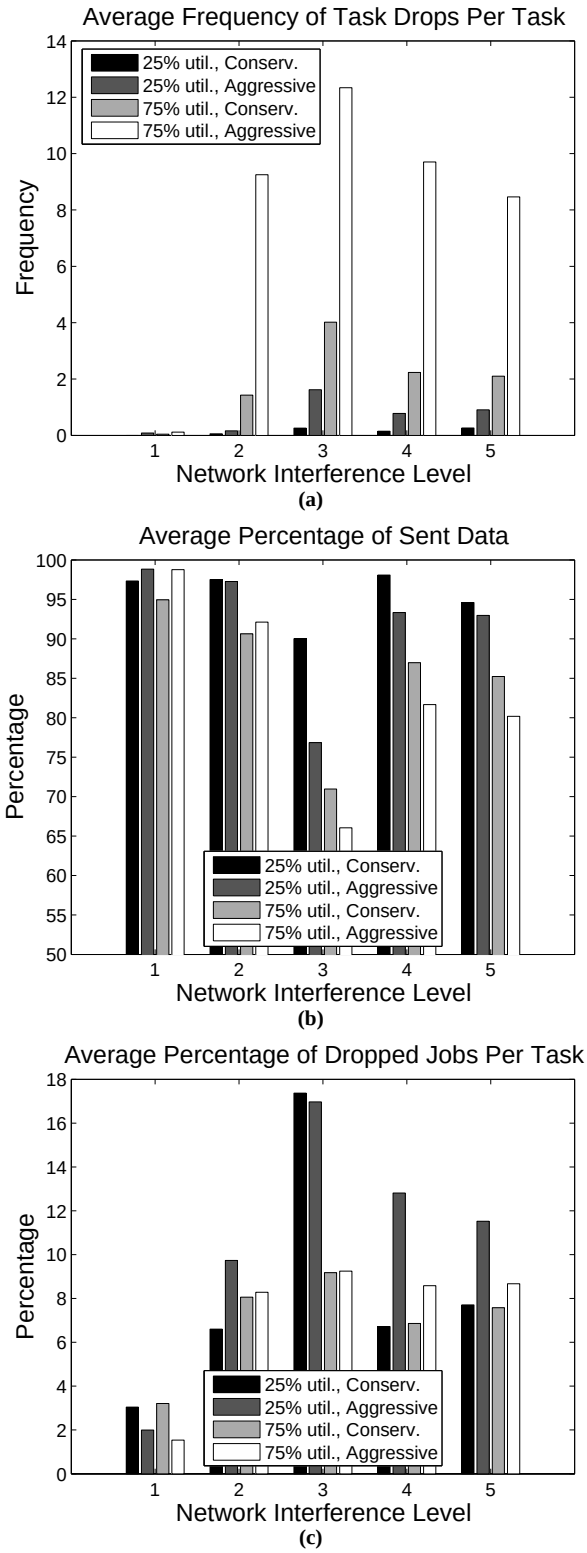


Figure 2: Statistics on **(a)** task drops per task; **(b)** percentage of data sent; and **(c)** percentage of dropped jobs per task.

less frequently; however, this can result in both a highly-underutilized system and trouble maintaining sending rates, as can be observed in inset **(b)**. Finally, note that performance is worse along all metrics in Fig. 2 at 75% utilization except for the number of dropped jobs. Again, this may be due to the fact that the number of dropped *tasks* is vastly higher at 75% versus 25% utilization.

We next determine both energy savings and the (m, k)-firm guarantees that we can make. Fig. 3 shows the (m, k)-firm guarantees for various task periods and values of k, given the feedback parameters that provide the best such guarantee at each interference level and system utilization; this should help to determine if Theorem 2 is useful in practice. The energy savings resulting from using these parameters is also shown, and compared to the most energy-efficient parameters. In many cases, the increased energy savings from parameters that resulted in a less-stable network is due to an increase in dropped tasks or jobs, which reduces the amount of data that must be sent and thus the amount of energy used.

First, note from insets **(a)** and **(b)** that the energy savings is significant in all scenarios—in some cases, the savings is quite dramatic. This remains true when comparing the energy savings of the parameters that produce the best (m, k)-firm results to those that achieve the greatest energy savings (typically at the detriment of other performance metrics). Second, while the 802.11 radio experienced the greatest energy savings at lower levels of interference and lower utilizations, both absolute and relative to the most energy-efficient parameters, the results seem to be reversed for the 802.15.4 radio. The 802.11 result is more intuitive—feedback parameters have the least impact on performance when interference is low, and can result in many dropped tasks that result in an increase in energy savings when interference is high. Additionally, it is much easier to save energy by turning off the radio when system utilization is low.

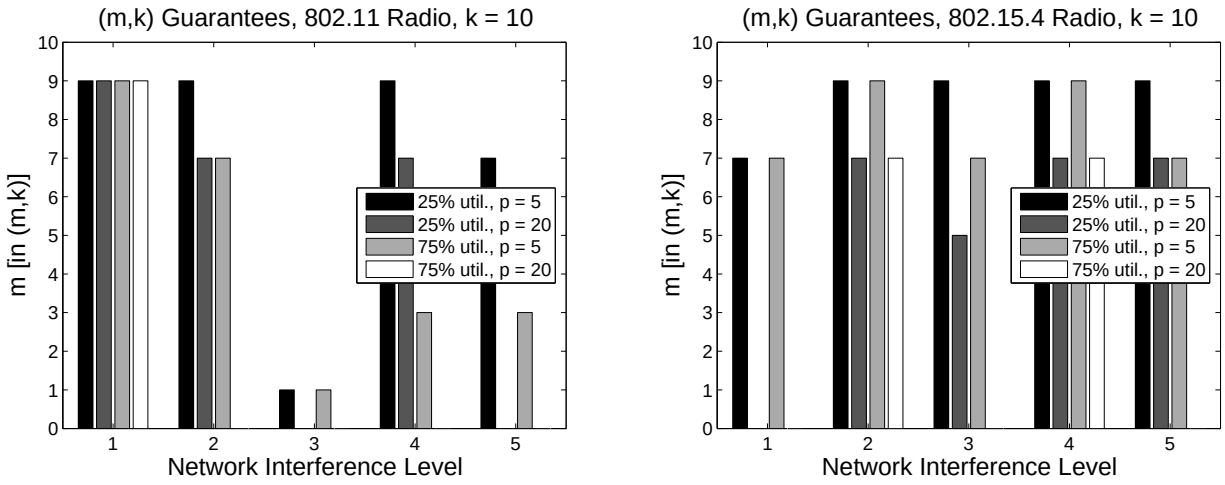
For the 802.15.4 radio, our reverse results might be explained by considering the impact of the 802.11 radio on the 802.15.4 radio. At high system utilizations, admission control is more likely to determine that placing most of the workload on the 802.11 radio is most energy-efficient. This is particularly true when interference is high, because tasks will be dropped more frequently, allowing for migrations between radios to occur when it is energy-efficient to do so. Since tasks have relatively high utilizations on the 802.15.4 radio as compared with the 802.11 radio, dropped tasks are much more likely to be re-admitted onto the 802.11 radio, especially during periods of high interference. Therefore, high system utilizations may actually result in a *lower* 802.15.4 radio utilization, thus producing the counter-intuitive result. Further, interference may cause migrations that force tasks off of the 802.15.4 radio and onto the 802.11 radio, thus reducing the opportunity for energy savings on that radio, but increasing energy savings on the 802.15.4 radio. In the case where no interference exists, task drops will occur less frequently and will typically result in bandwidth fluctuations

802.11 Radio					
Int. Level	**Util.**	**Best (m,k) Feedback Params.**	**Avg. Energy Saved (best (m,k) result)**	**Avg. Energy Saved (most efficient)**	**% Relative Savings vs. Most Efficient**
1	25%	(any)	159.686 J / 71.07%	71.07%	100.00%
1	75%	100 inc, (any) dec, (any) min int	74.61 J / 33.32%	41.11%	81.04%
2	25%	100 inc, 2 dec, 10 min int	129.16 J / 57.55%	63.19%	91.07%
2	75%	100 inc, 4 dec, 10 min int	36.67 J / 16.41%	25.82%	63.55%
3	25%	100 inc, 4 dec, 10 min int	47.54 J / 43.54%	60.23%	72.28%
3	75%	100 inc, 4 dec, 10 min int	30.10 J / 14.98%	21.43%	69.91%
4	25%	100 inc, 4 dec, 100 min int	107.57 J / 50.17%	64.02%	78.36%
4	75%	100 inc, 4 dec, 10 min int	31.15 J / 14.62%	26.79%	54.57%
5	25%	100 inc, 2 dec, 1 min int	115.83 J / 51.67%	63.44%	81.45%
5	75%	100 inc, 4 dec, 1 min int	33.28 J / 15.57%	26.66%	58.38%

(a)

802.15.4 Radio					
Int. Level	**Util.**	**Best (m,k) Feedback Params.**	**Avg. Energy Saved (best (m,k) result)**	**Avg. Energy Saved (most efficient)**	**% Relative Savings vs. Most Efficient**
1	25%	500 inc, 2 dec, 10 min int	81.92 mJ / 15.26%	31.75%	48.08%
1	75%	100 inc, 2 dec, 1 min int	152.17 mJ / 25.32%	41.14%	61.55%
2	25%	100 inc, 2 dec, 100 min int	27.36 mJ / 16.80%	50.16%	33.50%
2	75%	500 inc, 2 dec, 100 min int	169.33 mJ / 23.43%	42.15%	55.58%
3	25%	100 inc, 2 dec, 10 min int	168.00 mJ / 24.41%	26.88%	90.81%
3	75%	100 inc, 4 dec, 1 min int	252.27 mJ / 29.52%	29.52%	100.00%
4	25%	100 inc, 4 dec, 1 min int	54.90 mJ / 36.16%	43.16%	83.78%
4	75%	100 inc, 4 dec, 100 min int	213.02 mJ / 38.60%	43.81%	88.10%
5	25%	100 inc, 4 dec, 1 min int	139.15 mJ / 30.46%	36.10%	84.37%
5	75%	100 inc, 4 dec, 1 min int	194.55 mJ / 26.70%	36.57%	73.02%

(b)



(c) **(d)**

Figure 3: (m,k)-firm guarantees with the best feedback parameters for each scenario, and energy savings for those parameters.

that do not force task migrations, and as a result, the relative energy savings in those cases is lower.

In terms of stability guarantees, the best guarantees can made when task periods are small, system utilization is low, and interference is low (see insets **(c)** and **(d)**). In the case of the 802.11 radio, heavy interference can make it difficult to make *any* meaningful guarantee—however, note that in most cases, we can make a guarantee that 50-90% of the jobs will complete successfully. One exception is in the case of no interference for the 802.15.4 radio. Again, this may be due to the fact that migrations are never forced due to throughput fluctuations, and thus, a slight over-estimate of bandwidth can impact a lot of tasks, since the tasks have relatively high utilization on the 802.15.4 radio, resulting in weaker (m,k)-firm guarantees.

Note in insets **(a)** and **(b)** that the feedback parameters that achieved the best (m,k)-firm guarantees were typically conservative, and in most cases the 10-second minimum stable interval provided the best results. The 100-second minimum interval was often too large for any stable interval to be observed—instead, we observed one long unstable interval. The one-second interval was best at high levels of interfer-

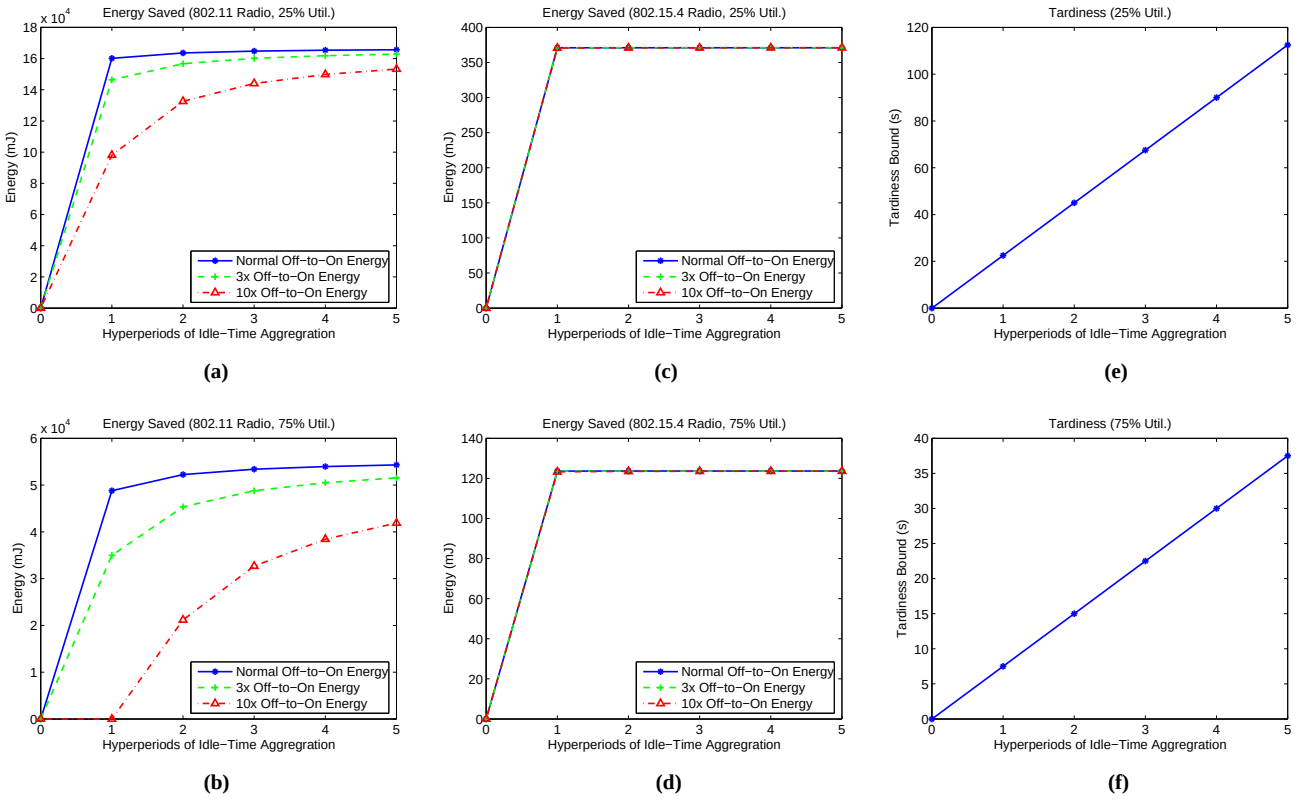


Figure 4: Energy savings and tardiness bounds resulting from idle-time aggregation, for 25% (top) and 75% (bottom) system utilization. The columns correspond (left to right) to 802.11 energy savings, 802.15.4 energy savings, and tardiness bounds.

ence and spikes, where large unstable intervals could occur unless we allowed smaller stable intervals.

6.2 Deadline Tardiness vs. Energy Savings

We now investigate the tradeoff stated in Theorem 1 between tardiness bounds and energy savings by calculating the energy saved per radio, and the resulting tardiness bounds, as we increase the number of hyperperiods over which we perform idle-time aggregation. The results for a 30-sec. hyperperiod and two different system utilizations are shown in Fig. 4, again assuming 300 seconds of run time. (Similar energy results were observed for other hyperperiod sizes; however, tardiness bounds will be different.) The off-to-on energy assumed was one, three, or ten times the actual value, which allowed us to see the results of extremely high off-to-on energy requirements. Note that, even at ten times normal off-to-on energy, we almost always observe a large energy savings after idle-time aggregation over one hyperperiod, and quickly diminishing returns thereafter, relative to the increase in tardiness bounds. For the 802.15.4 radio, this is particularly apparent—we see virtually zero additional savings after one hyperperiod. This implies that there is a little benefit to increasing the number of hyperperiods over which we perform idle-time aggregation once the aggregated idle time exceeds the minimum idle interval to achieve energy savings by turning off the radio, as was discussed earlier in Sec. 5. Note, however, that even if we have to perform idle-time aggregation over many hyperperiods in order to achieve the minimum size idle interval, we can still miss deadlines by at most the amount of idle time aggregated. For example, if a three-second idle interval is required to turn off the radio, and we achieve or exceed such an interval by aggregating idle time over twenty hyperperiods, then we can still miss deadlines by at most three seconds (or perhaps slightly more depending on the size of the idle interval generated). Ultimately, this method simply provides a way to turn off a radio to save energy, in exchange for bounded tardiness by an amount that is approximately the size of the required idle interval—the number of hyperperiods over which we aggregate idle time in order to achieve this interval is not as important.

7 Conclusion

We have presented an algorithm for real-time scheduling of tasks on multiple heterogeneous radios. Our method is local in that it does not require broad knowledge about the networking environment, and is therefore suited to scenarios where interference is unpredictable. We show that the method results in significant energy savings by allowing radios to be turned off in ways that still allow real-time guarantees to be made. We also show that energy savings can

sometimes be significantly increased if we are willing to accept bounded deadline tardiness, but that such a method results in diminishing returns in energy savings as compared with the rate at which tardiness grows. We also propose a feedback mechanism that has been shown to effectively handle network interference, and with the appropriate feedback parameters, sufficient network stability is achieved to allow (m, k)-firm guarantees to be made in most cases. The evaluation confirmed that the analytical results presented in this paper are useful in practice.

There are several interesting areas of future work. First, we would like to determine the feasibility of using both radios to send data for the same job in parallel, so that job splitting could be used to alleviate task bin-packing issues. (This would make the multiprocessor scheduling problem fundamentally easier.) Second, we would like to explore different feedback mechanisms that improve the accuracy of our measurements and overall network stability. Third, we want to experiment with more dynamic real-time workloads. Fourth, we would like to explore methods for minimizing energy consumption while taking into account other factors such as the monetary cost of using a particular radio, or user preferences; this could perhaps be accomplished through the specification of an objective function. Finally, we would like to investigate the performance impact of more sophisticated energy accounting during admission control, such as simulating scheduling out to a hyperperiod.

References

[1] J. Anderson and J. Calandrino. Parallel real-time task scheduling on multicore platforms. *Proc. of the 27th IEEE Real-time Sys. Symp.*, pp. 89–100, 2006.

[2] J. Anderson and A. Srinivasan. Mixed Pfair/ERfair scheduling of asynchronous periodic tasks. *Journal of Computer and System Sciences*, 68(1):157–204, 2004.

[3] H. Aydin, V. Devadas, and D. Zhu. System-level energy management for periodic real-time tasks. *Proc. of the 27th IEEE Real-time Sys. Symp.*, 2006.

[4] B. Bui, R. Pellizzoni, M. Caccamo, C. Cheah, and A. Tzakis. Soft real-time chains for multi-hop wireless ad-hoc networks. *Proc. of the 13th IEEE Real Time and Embedded Technology and Applications Symp.*, pp. 69–80, 2007.

[5] H. Chen, B. Tjaden, L. Welch, C. Bruggeman, L. Tong, and B. Pfarr. Monitoring network QoS in a dynamic real-time system. *Proc. of the 16th Intl. Parallel and Distributed Processing Symp.*, 2002.

[6] D. Demarch and L. Becker. An integrated scheduling and retransmission proposal for firm real-time traffic in IEEE 802.11e. *Proc. of the 19th Euromicro Conf. on Real-Time Sys.*, pp. 146–158, 2007.

[7] S. Funk and S. Baruah. Task assignment on uniform heterogeneous multiprocessors. *Proc. of the 17th Euromicro Conf. on Real-Time Sys.*, pp. 219–226, 2005.

[8] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m,k)-firm deadlines. *IEEE Trans. on Computers*, 44(12):1443–1451, 1995.

[9] HTC 8525 smartphone specifications. `http://www.america.htc.com/products/8525/default.html`.

[10] V. Jacobson. Congestion avoidance and control. *Proc. of SIGCOMM '88*, pp. 314–320, 1988.

[11] K. Karenos and V. Kalogeraki. Real-time traffic management in sensor networks. *Proc. of the 27th IEEE Real-Time Sys. Symp.*, pp. 422–434, 2006.

[12] A. Koubaa, M. Alves, and E. Tovar. Modeling and worst-case dimensioning of cluster-tree wireless sensor networks. *Proc. of the 27th IEEE Real-Time Sys. Symp.*, pp. 412–421, 2006.

[13] H. Leontyev and J. Anderson. Generalized tardiness bounds for global multiprocessor scheduling. *Proc. of the 28th IEEE Real-time Sys. Symp.*, 2007. To appear.

[14] C. Lu, J. Stankovic, G. Tao, and S. Son. Design and evaluation of a feedback control EDF scheduling algorithm. *Proc. of the 20th IEEE Real-Time Sys. Symp.*, 1999.

[15] C. Lu, J. Stankovic, G. Tao, and S. Son. Feedback control real-time scheduling: Framework, modeling and algorithms. *Real-Time Sys. Journal, Special Issue on Control-Theoretical Approaches to Real-Time Computing*, 23(1/2):85–126, 2002.

[16] C. Lu, X. Wang, and X. Koutsoukos. Feedback utilization control in distributed real-time systems with end-to-end tasks. *IEEE Trans. on Parallel and Distributed Sys.*, 16(6):550–561, 2005.

[17] D. Lymberopoulos, B. Priyantha, and F. Zhao. mPlatform: A reconfigurable architecture and efficient data sharing mechanism for modular sensor nodes. *Proc. of the Intl. Conf. on Information Processing in Sensor Networks*, 2007.

[18] D. McIntire, K. Ho, B. Yip, A. Singh, W. Wu, and W. Kaiser. The low power energy aware processing (LEAP) embedded networked sensor system. *Proc. of the 5th Intl. Conf. on Information Processing in Sensor Networks*, pp. 449–457, 2006.

[19] T. Pering, Y. Agarwal, R. Gupta, and R. Want. CoolSpots: Reducing the power consumption of wireless mobile devices with multiple radio interfaces. *Proc. of the 4th Intl. Conf. on Mobile Sys., Applications and Services (MobiSys)*, pp. 220–232, 2006.

[20] K. Prabh and T. Abdelzaher. On scheduling and real-time capacity of hexagonal wireless sensor networks. *Proc. of the 19th Euromicro Conf. on Real-Time Sys.*, pp. 136–145, 2007.

[21] W. Qadeer, T. Rosing, J. Ankcorn, V. Krishnan, and G. Micheli. Heterogeneous wireless network management. *Proc. of the Third Intl. Workshop on Power-Aware Computer Sys.*, pp. 86–100, 2003.

[22] E. Shih, P. Bahl, and M. Sinclair. Wake on wireless: An event driven energy saving strategy for battery operated devices. *Proc. of the Eighth ACM Intl. Conf. on Mobile Computing and Networking*, 2002.

[23] T. Stathopoulos, M. Lukac, D. McIntire, J. Heidemann, D. Estrin, and W. Kaiser. End-to-end routing for dual-radio sensor networks. *Proc. of the IEEE Infocom*, 2007.

[24] X. Wang, D. Jia, C. Lu, and X. Koutsoukos. DEUCON: decentralized end-to-end utilization control for distributed real-time systems. *IEEE Trans. on Parallel and Distributed Sys.*, 18(6):1–14, 2007.