Network-Aware, Energy-Conscious, Fair Service for Real-Time Applications on Multiprocessor SoC[*]

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

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

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

Abstract

We consider systems consisting of wireless nodes that execute CPU intensive applications on multiprocessor system-on-a-chip (MPSoC) and must transmit packets over the network in a timely manner. Existing methods do not consider packet deadlines in conjunction with energy and real-time task performance, making it hard to predict system behavior. We present an energy-aware adaptive CPU scheduling algorithm to maximize the number of packet deadlines met in a fair manner and discuss future work.

1 Introduction and Related Work

Wireless networks are now common in a variety of applications, e.g., [3, 5]. While many wireless sensor nodes typically require minimum hardware to perform lightweight tasks (e.g., periodically waking up to sense and transmit data), powerful processing nodes can also be found in certain applications for executing computationally intensive tasks and transmitting packets over the network. Example applications are surveillance and mobile gaming systems. In a surveillance system, a wireless node periodically captures a video, processes the frames and transmits them to clients. As for the gaming system, the processor is kept busy with a large number of tasks (e.g., rendering graphics) while a large amount of data is sent to the user's opponents.

To cater to the high computing demand imposed by the applications mentioned above, one alternative is to use high-end, power-hungry, processors. Since wireless nodes are generally battery powered, to save energy and prolong the lifetime of these nodes, multiprocessor system-on-chips (MPSoCs) present a better alternative for wireless nodes requiring higher computational power. More work can be completed by running processor cores in parallel at lower voltage and frequency, thus saving energy. The parallel execution capabilities of such MPSoC-based wireless nodes, however, introduces new challenges in terms of reducing energy consumption while satisfying real-time constraints.

There is a large research body on energy minimization in multiprocessors running real-time applications, e.g., [1, 2], though the majority of the work solely focuses on optimizing real-time task performance without any consideration for packet deadlines. Meeting packet deadlines is important in ensuring performance requirements such as data freshness. At the same time, network-aware work usually focuses on trading network energy with packet latency using packet scheduling and ignores task deadlines [7, 8]. To the best of our knowledge, the work by Yi et al. is the only energy-aware solution that explicitly considers packet deadlines on uniprocessor architectures executing real-time tasks [9]. However, it is unclear how the proposed solution may extend to multiprocessor architectures. Additionally, in the approach of [9], it is difficult, if not impossible, to predict which packets will miss their deadlines. This could lead to unfairness in packet transmission. That is, some tasks may consistently be able to successfully transmit their packets while the packets of other tasks starve.

In this work, we design an adaptive CPU scheduling algorithm that maximizes the number of packets that meet their deadlines in a fair manner. Fairness is used to ensure that each task has an adequate number of successfully transmitted packets relative to their importance. The main idea of our work is to prevent executions of jobs whose packets will be dropped to save CPU energy while allowing specific packets to be sent. Using a network reservation-based approach in [9], network energy is also managed.

2 Preliminaries

We consider a set of n independent periodic real-time tasks. Each task τ_i is described by its worst-case execution

[*]This work is supported in part by NSF under grant numbers CNS-0834180 and CNS-0834230.

time C_i and period T_i. All tasks are synchronous. The j-th instance (job) of task τ_i is denoted by $\tau_{i,j}$. We assume a partitioned scheduling approach in which tasks are assigned to their respective cores using the algorithm in [2] and that the tasks on each core are schedulable using the algorithm in [6]. No job or task migration is allowed.

Two types of tasks are considered: packet-generating and non packet-generating. Non packet-generating tasks are hard real-time tasks. Without loss of generality, we assume that every job of packet-generating tasks generates a packet at the end of its execution. Packets have firm real-time deadlines, i.e., they must be transmitted by their deadline or they will be dropped. Packets from different instances of the same task are equal in size while packets from different tasks may vary in size. A packet P_j is described by its deadline X_j and worst-case transmission time Z_j.

The MPSoC consists of m homogeneous cores, each of which can run at k discrete speed levels. Cores can independently change speed. We assume that transition overheads associated with switching from one speed to another have been included in the task worst-case execution times.

The processor cores share a network card. Packets are transmitted in an earliest-deadline first (EDF) manner. Transmissions are preemptable at some minimum unit. Since packets that cannot meet their deadlines are dropped, the number of packets transmitted is used interchangeably with the number of packets that meet their deadlines.

Each node uses TDMA-like periodic time slots to send and receive packets. No network communication takes place outside of these time slots. The time slots, described by a period T_{ts} and length C_{ts}, may change over time to reflect different network usage levels. We assume that incoming packets are buffered at the sender and arrive at the beginning of each time slots during which the wireless node avoids transmitting any packets.

We are interested in solving the following problem: Given the real-time task set, MPSoC, network card and transmission models described above, determine an execution pattern for packet-generating jobs such that all non packet-generating jobs meet their deadlines, the number of packets transmitted over the network is maximized in a fair manner, and the energy consumption is minimized.

3 Motivations

We use a simple example to motivate our problem. Assume that we have a set of four tasks as shown in Table 1. In addition, our MPSoC consists of four identical cores, m_1, m_2, m_3, and m_4. Using the approximation algorithm in [2], each core is assigned a task to execute.

For this example, we assume that jobs always require their worst-case execution times, that cores can adjust their speeds in a continuous manner, and that the network card is

Table 1. Example Task Set

Task	C	T	Packet?	X	Z
τ_1	1	2	Yes	$T+1$	0.5
τ_2	1	3	Yes	$T+2$	1
τ_3	2	6	Yes	$T+2.3$	1
τ_4	3	12	Yes	$T+2.3$	1

allowed to transmit packets whenever it wishes. It must be noted, however, that these assumptions are made for ease of explanation and that we do not rely on any such assumption to solve our general problem. Using LaEDF [6], each job finishes its execution right before its deadline. The corresponding processor and network card states are shown in Figures 1(a) and 1(b), respectively.

As shown in Figure 1(b), no packets generated by tasks τ_3 and τ_4 were transmitted. This is unfair because all packets generated by tasks τ_1 and τ_2 were sent. What is worse, the same pattern will persist to the future, which means that packets generated by τ_3 and τ_4 will never be transmitted and the energy used to execute instances of τ_3 and τ_4 is wasted.

Now, suppose that we had a mechanism to select jobs to execute in a fair manner. The resultant MPSoC and network card state might be as shown in Figures 2(a) and 2(b), respectively. The total number of packets transmitted does not change but some packets generated by τ_3 and τ_4 can now be transmitted. In addition, the CPU does not waste energy executing jobs whose packets are not transmitted.

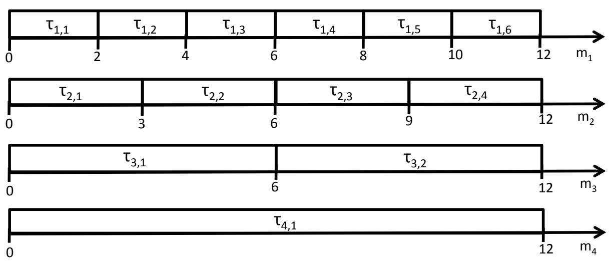
4 Network-Aware Adaptive CPU Scheduling

We provide our general approach, discuss the fairness metric, and give a detailed explanation of our algorithm.

4.1 Overview

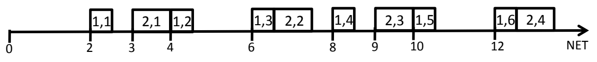
The main component of our algorithm is the manager task (MT), which runs on one of the cores, is a periodic task, and competes for resource. Let us define an observation window (ObsWin) to be the least common multiple of the task hyperperiod and T_{ts}. In an ObsWin, the network card monitors the packet transmission pattern and sends this information to the MT, which will then use it to compute the current system fairness level. The MT also compiles a list of tasks whose packets are being transmitted more than it should and send it to the cores. In the next ObsWin, the scheduler on each core avoids executing jobs of these tasks during high interference time intervals (defined below), thus allowing packets from other tasks to be transmitted.

Since cores execute jobs using dynamic voltage scaling (DVS) and some jobs are dropped, processor energy is saved. At the same time, since these jobs are dropped in a controlled manner, packets will be transmitted fairly. Note



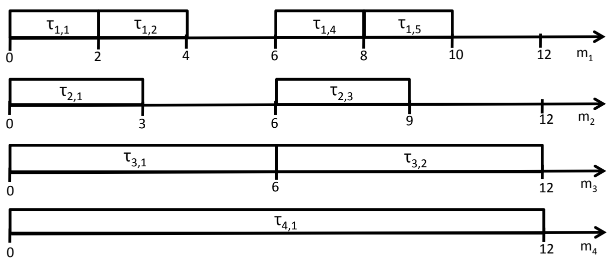
$\tau_{1,1}$ $\tau_{1,2}$ $\tau_{1,3}$ $\tau_{1,4}$ $\tau_{1,5}$ $\tau_{1,6}$
0 2 4 6 8 10 12 m_1
$\tau_{2,1}$ $\tau_{2,2}$ $\tau_{2,3}$ $\tau_{2,4}$
0 3 6 9 12 m_2
$\tau_{3,1}$ $\tau_{3,2}$
0 6 12 m_3
$\tau_{4,1}$
0 12 m_4

(a) Processor state.



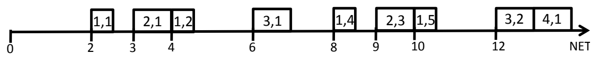
(b) Network state, each box indicates packet by $\tau_{i,j}$ is transmitted.

Figure 1. System state without network-aware scheduling.



$\tau_{1,1}$ $\tau_{1,2}$ $\tau_{1,4}$ $\tau_{1,5}$
0 2 4 6 8 10 12 m_1
$\tau_{2,1}$ $\tau_{2,3}$
0 3 6 9 12 m_2
$\tau_{3,1}$ $\tau_{3,2}$
0 6 12 m_3
$\tau_{4,1}$
0 12 m_4

(a) Processor state.



(b) Network state.

Figure 2. System state with network-aware scheduling.

that we focus on CPU energy and not network energy, as the network energy has already been managed during the negotiation of the periodic time slots [9].

Our algorithm is adaptive in the sense that it may take a number of ObsWins for the packet transmission pattern to stabilize. However, the system can perform exactly as it has in the previous ObsWin once this is the case until, say, a new task joins the system or the period or length of the time slot changes. In such cases, the MT is reactivated until the system stabilizes yet again.

4.2 Fairness Metric

Although our algorithm is independent of the specific fairness metric used, we use Jain's fairness index [4] to measure system fairness level in this work. Jain's fairness index F can be defined as follows.

$$F = \frac{\left(\sum_{\tau_i \in G}(w_i \lambda_i)\right)^2}{|G| \cdot \sum_{\tau_i \in G}(w_i \lambda_i)^2} \leq 1 \qquad (1)$$

where w_i is the weight of task τ_i, G is the set of packet-generating tasks, and λ_i denotes the packet deadline meet ratio of packet-generating task τ_i.

4.3 Algorithm Details

The relevant parts of our proposed algorithm are shown in Algorithm 1, with checkpoints omitted for brevity.

To observe the original system behavior during the first ObsWin (e.g., where packet deadline misses occur, if any), all jobs are executed using LaEDF and the network card transmits as many packets as possible while keeping track of the deadline miss ratios $\boldsymbol{\lambda}$, as well as $\mathbf{HT}$, which is a list of high interference time intervals $[t_s, t_e]$ where t_s is the release time of a packet whose deadline is missed and t_e is its deadline. If there is no packet deadline miss, our algorithm is not activated and the system continues as before.

Using $\boldsymbol{\lambda}$, the MT computes F as in (1). The objective of the MT is to improve the system fairness level by identifying tasks which have had an unfair access to the network. That is, we wish to reduce the number of packets transmitted by such tasks so that the number of packets transmitted by other tasks (which have been disadvantaged) can be increased. Specifically, we use two embedded loops that iterate through packet-generating tasks, one following the non-increasing order of meet ratios and the other in reverse. The goal is to find pairs of tasks, one with a higher meet ratio and another with a lower meet ratio, whose packets will likely interfere with one another. The MT then determines how the system fairness level will change should the number of packets of the task with a higher (lower) meet ratio is decreased (increased) by one. If the system fairness level will increase, the task with the higher meet ratio is added to the set $\mathbf{TL}$, which will be used by the scheduler on each core to determine which jobs to drop.

As for the scheduler on each core, its only duty is to determine whether to execute the next ready job. This is accomplished by examining whether the associated task of the current job appears in $\mathbf{TL}$ and determining whether the job deadline falls within one of the high interference time intervals. Here, the idea is to balance energy consumption with useful work completed by the processor cores.

5 Summary and Future Work

We proposed an energy-aware adaptive CPU scheduling algorithm to maximize the number of packets transmitted

Algorithm 1 Energy-Aware Adaptive CPU Scheduling

```
Upon end of each ObsWin
  if at least one packet missed its deadline then
    execute MT with $\boldsymbol{\lambda}$ and $\mathbf{HT}$ from network card
Upon each execution of MT
  compute $F$ // current fairness level of system
  $R \leftarrow$ packet-generating tasks sorted in a non-increasing order of $\mathbf{w}\boldsymbol{\lambda}$
  $R' \leftarrow inverse(R)$
  for each task $\tau_i \in R$ do
    $\lambda_i \leftarrow \frac{\text{num\_packets\_transmitted}_i - 1}{\text{num\_packets\_generated}_i}$
    for each task $\tau_j$ in $R'$ do
      if $\tau_i$.deadline and $\tau_j$.deadline $\in \mathbf{HT}$ then
        $\lambda_j \leftarrow \frac{\text{num\_packets\_transmitted}_j + 1}{\text{num\_packets\_generated}_j}$
        compute $F'$ // using new values for $\lambda_i$ and $\lambda_j$
        if $F' > F$ then
          $\mathbf{TL} \leftarrow \mathbf{TL} \cup \tau_i$
          $F \leftarrow F'$
  send $TL(m)$ and $HT(m)$ to $m$, for all cores $m$
Upon each scheduling point on core m
  $j \leftarrow$ next ready job // using LaEDF
  if $j$.deadline $\in HT$ and $j$.getTask $\in TL$ then
    drop $j$
```

over the network in a fair manner for real-time applications running on MPSoCs. Our algorithm is independent of the fairness metric used and requires minimum interactions between the network card and the processor cores.

As this work is ongoing, there are still many improvements to be made and several challenges to be solved. For instance, the observation window is currently defined to be the least common multiple of the task hyperperiod and T_{ts}. While this definition of ObsWin allows the MT to easily determine which jobs should not execute and when, the actual length of ObsWin in practice may be too large since it is a function of the task hyperperiod. Using a large ObsWin has several drawbacks. For instance, changes may take place very slowly, prohibiting the system from responding to dynamic perturbations in a timely manner. Also, a long ObsWin entails that the size of $\mathbf{HT}$ will be large, requiring a large amount of memory space and causing long latency when executing our algorithm.

In the current version of the algorithm, some jobs are dropped even if their packets may in reality not interfere with the other packets because of the high variation in job execution times. In other words, the current algorithm may too aggressively drop jobs. A heuristic is needed to determine whether such jobs should be executed based on their expected execution times and network state. Also, job migration may help to further improve the system fairness level, as well as energy saving, and is worth exploring.

An implicit philosophy behind the proposed algorithm is the minimization of the interactions between the network card and processor cores, as a constant update from the network card regarding its status can incur unacceptable overheads. However, due to the lack of constant communications, the cores do not have full knowledge of the network state, which could greatly help in improving performance. At the same time, the use of the MT requires that both $\mathbf{TL}$ and $\mathbf{HT}$ be shared, possibly via cache. This means that our algorithm may require that the memory management unit arbitrates the access of these shared information (and possibly causing delays in some job executions when the MT is executing). Determining the "right" amount of status update from the network card and the best way to share information among cores are subject to ongoing investigation.

Finally, once all the above challenges have been addressed, we plan on evaluating our work against the work in [9] for uniprocessor architectures and against the state-of-the-art energy-aware algorithm that does not consider packet deadlines for MPSoC cases.

References

[1] H. Aydin and Q. Yang. Energy-aware partitioning for multiprocessor real-time systems. In *Proc. of the Int. Symp. on Parallel and Distributed Processing*, page 113.2, Apr. 2003.

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

[3] T. H. et al. VigilNet: An integrated sensor network system for energy-efficient surveillance. *Trans. on Sensor Networks*, 2(1):1–38, Feb. 2006.

[4] R. Jain, D. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Technical report, DEC Research Report, 1984.

[5] A. Mainwaring, J. Polastre, R. S. D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proc. of the Int. Workshop on Wireless Sensor Networks and Applications*, pages 88–97, Sept. 2002.

[6] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. *Operating Systems Review*, 35(5):89–102, Oct. 2001.

[7] V. Raghunathan, S. Ganeriwal, C. Schurgers, and M. Srivastava. E2WFQ: An Energy Efficient Fair Scheduling Policy for Wireless Systems. In *Proc. of the Int. Symp. on Low Power Electronics & Design*, page 30.

[8] E. Uysal-Biyikoglu, B. Prabhakar, and A. E. Gamal. Energy-efficient packet transmission over a wireless link. *Trans. on Networking*, 10(4):487–499, Aug. 2002.

[9] J. Yi, C. Poellabauer, X. Hu, J. Simmer, and L. Zhang. Energy-conscious co-scheduling of tasks and packets in wireless real-time environments. In *Proc. of the Real-Time and Embedded Technology and Applications Symp.*, pages 265–274, Apr. 2009.