Optimal Semi-Partitioned Scheduling in Soft Real-Time Systems *

James H. Anderson[1], Jeremy P. Erickson[1], UmaMaheswari C. Devi[2], and Benjamin N. Casses[1]
[1]Dept. of Computer Science, University of North Carolina at Chapel Hill
[2]IBM Research - India

Abstract

Semi-partitioned real-time scheduling algorithms extend partitioned ones by allowing a (usually small) subset of tasks to migrate. The first such algorithm to be proposed was directed at soft real-time (SRT) sporadic task systems where bounded deadline tardiness is acceptable. That algorithm, called EDF-fm*, has the desirable property that migrations are* boundary-limited*, i.e., they can only occur at job boundaries. However, it is not optimal because per-task utilization restrictions are required. In this paper, a new optimal semi-partitioned scheduling algorithm for SRT sporadic task systems is proposed that eliminates such restrictions. This algorithm, called* EDF-os*, preserves the boundary-limited property. In overhead-aware schedulability experiments presented herein,* EDF-os *proved to be better than all other tested alternatives in terms of schedulability in almost all considered scenarios. It also proved capable of ensuring very low tardiness bounds, which were near zero in most considered scenarios.*

1 Introduction

Multiprocessor real-time scheduling algorithms may follow a *partitioned* or *global* approach or some hybrid of the two. Under partitioned scheduling, tasks are statically assigned to processors, while under global scheduling, they are scheduled from a single run queue and hence may migrate. When comparing different scheduling approaches, one criterion is *optimality*, i.e., the ability to correctly schedule (without timing-constraint violations) any task system for which a correct schedule exists. In the case of implicit-deadline (see Sec. 2) sporadic task systems, optimality can be achieved via global scheduling, but not partitioning; however, global scheduling entails higher runtime overheads. When designing a hybrid approach, the goal is usually to attain optimal or near-optimal behavior but with less overhead than a truly global approach.

One such hybrid approach is *semi-partitioned* scheduling, which extends partitioned scheduling by allowing those tasks that cannot be feasibly assigned to processors to migrate. Semi-partitioned scheduling was first proposed for supporting implicit-deadline soft real-time (SRT) sporadic task systems under the "bounded deadline tardiness" definition of SRT (which is the definition of SRT we assume hereafter) [3]. Subsequently, several semi-partitioned algorithms were proposed for hard real-time (HRT) systems where no deadline misses are tolerable [5, 6, 11, 12, 13, 16, 18, 21, 22, 23, 24, 25, 26, 27, 28, 32, 33]. In these prior efforts, various migration strategies for migrating tasks were proposed. Given the stated goal of "less overhead," it is desirable for such a strategy to be *boundary-limited*, i.e., to allow a migrating task to migrate only between job boundaries (i.e., between successive invocations). Non-boundary-limited schedulers allow *jobs* to migrate, which can be expensive in practice if jobs maintain much cached state.

Of the algorithms cited in the prior paragraph, only one, namely EKG [6], meets the stated goal of optimality for the class of systems for which it was designed, and only in a very restricted sense: it is optimal only for periodic (not sporadic) task systems, and only when a configurable parameter k becomes equal to the number of processors, which unrealistically increases preemption frequency. Moreover, EKG is not boundary-limited. In fact, it can be easily shown that no boundary-limited scheduler can be optimal for HRT sporadic task systems.

In this paper, we show that a semi-partitioned scheduling algorithm that is both optimal and boundary-limited does exist in the SRT case. We use the usual definition of the term "optimal," as given earlier; however, for SRT schedulers as considered here, the phrase "correctly schedule" means that *bounded tardiness* can be guaranteed for each task. Under this definition, prior work shows that correct scheduling is possible via global scheduling if no task over-utilizes a single processor and the entire system of processors is not over-utilized [17]. No further utilization restriction is necessary. In the context of global scheduling, the boundary-limited property can be achieved by executing jobs non-preemptively, which increases tardiness bounds. This property can be sacrificed, yielding smaller bounds, by executing jobs preemptively. However, overheads then become a problem. In this regard, semi-partitioned schedulers have a key advantage over global ones as the former use *push migrations*, which are pre-planned, while the latter use *pull migrations*, which are reactive in nature and thus more difficult to account for and implement efficiently [10].

The original SRT algorithm from [3], called EDF-fm, is boundary-limited and able to fully utilize the underlying hardware platform's available capacity. However, it requires per-task utilization restrictions that render it non-optimal. In their simplest form, these restrictions preclude any task utilization from exceeding half the capacity of a processor, though they can be relaxed somewhat, as discussed below.

Contributions. In this paper, we close a problem that has stood open since the initial publication of EDF-fm in 2005 [2] by presenting the first optimal semi-partitioned scheduling algorithm for SRT sporadic task systems.

*Work supported by NSF grants CNS 1016954, CNS 1115284, CNS 1218693, and CNS 1239135; and ARO grant W911NF-09-1-0535.

This algorithm, called EDF-os (earliest-deadline-first-based optimal semi-partitioned scheduling), is boundary-limited, like EDF-fm. However, it eliminates the need for EDF-fm's per-task utilization restriction through the use of several novel techniques.

Like any semi-partitioned algorithm, EDF-fm functions in two phases: an offline *assignment phase*, where tasks are assigned to processors and *fixed* tasks (which do not migrate) are distinguished from *migrating* ones (which do); and an online *execution phase*. In EDF-fm's execution phase, rules that extend earliest-deadline-first (EDF) scheduling are used to execute fixed and migrating tasks. Specifically, each migrating task executes on two processors, and for each processor, at most two specific migrating tasks may execute upon it. The tardiness-bound proof for EDF-fm relies crucially on the fact that migrating tasks never miss deadlines. To ensure this, migrating tasks are statically prioritized over fixed ones, jobs of migrating tasks are prioritized against each other on a EDF basis, and two migrating tasks that may execute on the same processor are limited to have a combined utilization of at most one. This last requirement restricts per-task utilizations.

In EDF-os, we eliminate unnecessary utilization restrictions by altering both phases. The assignment phase of EDF-os differs from that of EDF-fm in that we consider tasks in a specific order and allow a migrating task to execute on any number of processors (instead of just two). The execution phase of EDF-os differs from that of EDF-fm in that we statically prioritize jobs of certain migrating tasks over those of others, instead of prioritizing them against each other on an EDF basis. As a result of our different strategies, migrating tasks can miss deadlines. However, we show that tardiness bounds can be derived by leveraging certain properties pertaining to our EDF-os assignment phase and the more predictable nature of the static prioritizations we introduce. These properties allow us to apply a novel *reduction method* that enables a given system to be converted to a simpler form that can be more easily analyzed. This analysis shows that EDF-os is optimal, regardless of whether deadlines are implicit, constrained, or unrestricted (see Sec. 2). In contrast, the analysis for EDF-fm is restricted to implicit-deadline systems.

To evaluate EDF-os, we conducted an experimental evaluation in which it was compared to other algorithms, including EDF-fm. In this evaluation, the effects of measured overheads from actual operating-system-based scheduler implementations were factored into schedulability and tardiness analysis. From a schedulability standpoint, i.e., the ability to guarantee bounded tardiness, EDF-os proved to be the best algorithm in almost all considered scenarios. The magnitude of such bounds is important as well. In this regard also, EDF-os excelled, ensuring very low bounds generally, and near-zero bounds in most considered scenarios.

Organization. We present our optimality proof (Sec. 4) after first providing needed background (Sec. 2) and describing EDF-os in detail (Sec. 3). We then present our experimental evaluation (Sec. 5) and conclude (Sec. 6).

2 Background

We consider the scheduling of a sporadic task system $\tau = \{\tau_1, \tau_2, \ldots, \tau_N\}$ on M identical processors, $P_1, \ldots, P_M$ (we assume familiarity with the sporadic and periodic task models). Task τ_i is specified by (C_i, T_i), where C_i is its maximum[1] per-job execution requirement and T_i is its period. The j^{th} job of τ_i, denoted $\tau_{i,j}$, has release time $r_{i,j}$ and deadline $d_{i,j}$. We initially restrict attention to *implicit* deadlines ($d_{i,j} = r_{i,j} + T_i$) but in an online appendix [4] we consider both *constrained* deadlines ($d_{i,j} \leq r_{i,j} + T_i$) and *unrestricted* deadlines (no relationship between $d_{i,j}$ and $r_{i,j} + T_i$ assumed). We denote the *utilization* of τ_i by $U_i = C_i/T_i$, and the p^{th} processor as P_p. We assume that time is discrete.

In the scheduling algorithms we consider, each task is allocated a non-zero fraction, or *share*, of the available utilization of 1.0 on certain processors. Task τ_i's share (potentially zero) on P_p is denoted $s_{i,p}$. The total share allocation on P_p is denoted $\sigma_p \triangleq \sum_{\tau_i \in \tau} s_{i,p}$. We require that $\sigma_p \leq 1.0$ and that each task's total share allocation matches its utilization: $U_i = \sum_{k=1}^{M} s_{i,k}$. If τ_i has non-zero shares on multiple (only one) processor, then it is a *migrating* (*fixed*) task.

The scheduling algorithms we consider have the additional property that each job of each task τ_i executes on a specific processor. The *fraction* of τ_i's jobs (potentially zero) that execute on processor P_p is denoted $f_{i,p}$. Such fractions are commensurate with τ_i's share allocations:

$$f_{i,p} = \frac{s_{i,p}}{U_i}. \tag{1}$$

The lowest-indexed processor to which migrating task τ_i assigns jobs is called its *first processor*.

If a job $\tau_{i,j}$ completes at time t, then its *lateness* is $t - d_{i,j}$ and its *tardiness* is $max(0, t - d_{i,j})$. Observe that if a job's lateness is negative, then its tardiness is zero; otherwise, its lateness and tardiness are identical. We seek scheduling algorithms that ensure bounded tardiness: for each task, there is an upper bound on the tardiness of any of its jobs. We consider only *feasible* task systems that satisfy the following conditions.

$$\forall \tau_i \in \tau, U_i \leq 1 \text{ and } \sum_{\tau_i \in \tau} U_i \leq M. \tag{2}$$

Note that if a job is tardy, then the release time of the next job of the same task is unaltered; in such a case, consecutive jobs of the same task must execute in sequence (no parallelism). It follows that, if a task's tardiness is bounded, then its long-term processor share will be commensurate with its utilization. Also, per-job response times are bounded. (These are desirable properties of the SRT notion of correctness we employ.)

[1]Mills and Anderson [31] have shown that the worst-case execution times pertaining to our model can be viewed as operating-system-enforced *budgets* that can be provisioned on an average-case (or near-average-case) basis. Stochastic analysis pertaining to such a provisioning is layered "on top of" tardiness analysis pertaining to (deterministic) budget allocations.

EDF-fm. The EDF-os algorithm presented herein was obtained through key design changes to EDF-fm [3] that enable new analysis techniques. EDF-fm consists of *assignment* and *execution* phases. During the assignment phase, tasks are allocated shares offline. The employed procedure allocates processor utilization (as shares) to tasks by considering each processor and task in turn. If the currently considered processor P_p has sufficient unallocated utilization, then the currently considered task τ_i is assigned to it as a fixed task; otherwise, τ_i exhausts the remaining unallocated utilization of P_p and receives the rest of its needed allocation from P_{p+1}.

In the execution phase, released jobs are scheduled online without migration (i.e., each job executes on only one processor). The following prioritizations are used on each processor: migrating tasks are prioritized over fixed ones, and jobs of a given type (fixed or migrating) are prioritized against each other on an EDF basis. By the assignment procedure just described, at most two migrating tasks can have non-zero shares on a given processor. It is required that for any two such tasks, their combined utilization is at most 1.0. This ensures that such tasks do not miss deadlines (which is a crucial property in the tardiness analysis of EDF-fm).

To ensure that fixed tasks have bounded tardiness, it is important that no processor be overloaded in the long run. This can be ensured by employing a mechanism that ensures that, in the long run, each migrating task τ_i submits an appropriate fraction of its jobs to each of the two processors on which it executes. Such fractions are given by (1) and are maintained by leveraging results from Pfair scheduling [7]. Under Pfair scheduling, scheduling decisions are made for each time slot of length one quantum. Further, for any feasible implicit-deadline, synchronous, periodic task system, a task with utilization u receives between $\lfloor u \cdot t \rfloor$ and $\lceil u \cdot t \rceil$ quanta by the integral time instant t.

The above Pfair guarantee is used to assign the jobs of a migrating task τ_i that executes on processors P_q and P_{q+1} as follows. A schedule of two Pfair tasks T_q and T_{q+1} of total utilization 1.0 is conceptually maintained, where T_p ($p = q$ or $p = q + 1$) corresponds to execution on processor P_p and has a utilization of $f_{i,p}$. If, in a uniprocessor schedule of these two Pfair tasks, T_p is allocated time slot t, then the t^{th} job of the migrating task τ_i is assigned to processor P_p. Using this idea, the guarantee provided by Pfair scheduling ensures that the following property holds under EDF-fm.

Property 1. *In any schedule, out of the first n jobs of a migrating task τ_i, the number of jobs assigned to some processor P_p is between $\lfloor f_{i,p} \cdot n \rfloor$ and $\lceil f_{i,p} \cdot n \rceil$.*

In EDF-fm, the Pfair schedule is maintained only implicitly, via a formula provided in [3].

Ex. 1. We now give an example task system that shows that if the task utilization restriction of EDF-fm is violated, then migrating tasks may miss deadlines. Such misses invalidate the tardiness analysis given in [3]. Consider the system $\tau = \{(4, 6), (2, 3), (5, 6), (2, 3), (1, 2), (2, 3)\}$. Because $\sum_{\tau_i \in \tau} U_i = 4$, τ is feasible on four processors. Because all task utilizations but one exceed $\frac{1}{2}$, and the other utilization is exactly $\frac{1}{2}$, EDF-fm's utilization restriction will be violated regardless of the order in which tasks are considered for assignment on a four-processor system. For the listed order, with the assignment of tasks depicted in Fig. 1, we show that deadlines may be missed by migrating tasks on P_2. The Pfair-based mapping formula will assign odd-indexed jobs of τ_2 to P_1 and even-indexed jobs of τ_2 to P_2. For τ_3, $f_{3,2} = \frac{2}{3} \cdot \frac{6}{5} = \frac{4}{5}$. The mapping formula will assign the first four jobs in each consecutive group of five jobs of τ_3 to P_2. Fig. 3 shows the first 25 time units of execution on P_2 assuming deadline ties are broken in favor of τ_2. Note that each of the first four jobs of τ_3 misses its deadline.

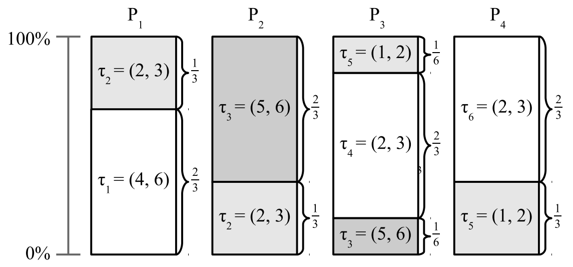

Figure 1: EDF-fm task assignment for Ex. 1.

3 EDF-os

In designing EDF-os, our goal was to eliminate EDF-fm's per-task utilization restrictions without altering the boundary-limited property or the fact that the underlying platform can be fully utilized. Because migrating tasks in EDF-fm cannot miss deadlines, tardiness under it can be analyzed on a per-processor basis. If such tasks *can* miss deadlines, then complex "couplings" of processors that are difficult to analyze arise: a miss by a migrating task on one processor can delay the processing of work due to it on another processor. Here, we show that such couplings can be dealt with by utilizing these key ideas:

- We use a worst-fit decreasing scheme to assign tasks to processors, rather than using an arbitrary ordering.
- Instead of prioritizing jobs of migrating tasks against each other using EDF, we statically give such a task the highest possible priority on any processor that is not its first processor (recall Sec. 2).
- By exploiting both modifications, we show that a reduction method can be used to analyze tardiness bounds in a simpler system in which complex "couplings" are eliminated.

EDF-os's assignment phase is described by the procedure in Fig. 4. An assignment is produced in two steps: first, as many tasks as possible are assigned as fixed tasks, using a worst-fit decreasing bin-packing heuristic. Then, all remaining tasks are assigned (in decreasing utilization order) by considering each processor and remaining task in turn. Each task considered in this step is allocated non-zero shares from a succession of processors until the sum of its shares equals its utilization. Because the remaining tasks are

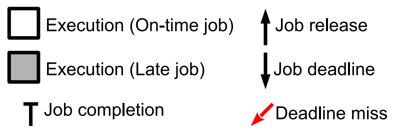

Figure 2: Key for Figures 3 and 6.

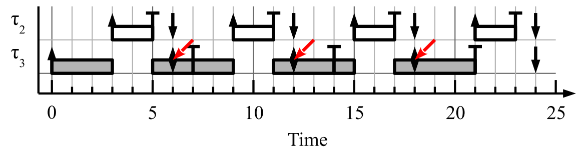

Figure 3: An EDF-fm schedule for the task system in Ex. 1 showing execution on P_2. Jobs from $\tau_3 = (5, 6)$ complete late.

considered in decreasing-utilization order, it is possible that such a task receives a non-zero share on only one processor, in which case it is a fixed task; otherwise, it is migrating. Like the assignment procedure for EDF-fm, this procedure ensures that there are at most two migrating tasks with non-zero shares on any processor. However, a migrating task under EDF-os can have non-zero shares on more than two processors. In this respect, EDF-os is similar to the C=D algorithm [16]. Note that EDF-os does not impose any restrictions on task utilizations (other than (2)), so it is now possible that migrating tasks may be tardy. Note also that, because tasks are considered in decreasing utilization order, each processor must contain at least one fixed task with a utilization at least that of any migrating task.

In the execution phase, EDF-os works as follows. As in EDF-fm, each job executes on only one processor. The *prioritization rules* used are as follows.

- On any processor, migrating tasks are statically prioritized over fixed ones (like in EDF-fm).
- Fixed tasks are prioritized against each other using EDF (like in EDF-fm).
- If a processor has two migrating tasks τ_i and τ_{i+1}, assigned in this order, then τ_i is statically prioritized over τ_{i+1} (this differs from EDF-fm). That is, a migrating task executes with highest priority on any processor that is not its first processor.

Informally, *the last rule ensures that tardiness is "created" for a migrating task only on its first processor; on its other processors, one of its jobs will be tardy only if its predecessor job was also tardy*. In fact, any such job assigned to a non-first processor will be scheduled as soon as it is eligible (i.e., released and its predecessor finished). As we shall see in the tardiness-bound proof in Sec. 4, *this very predictable execution behavior for "non-first-processor" jobs can be leveraged to derive a lateness bound for all migrating tasks, and in turn a tardiness bound for all fixed tasks.*

Because a migrating task may execute on more than two processors under EDF-os, the Pfair-based job-assignment formula used by EDF-fm cannot directly be applied to EDF-os. However, the same idea of using Pfair concepts to determine job assignments can continue to be used. In particular, if a migrating task τ_i executes on n processors, then we can conceptually manage n Pfair tasks with total

```
initially $s_{i,p} = 0$ and $\sigma_p = 0$ for all $i$ and $p$
/* assign fixed tasks via a worst-fit decreasing packing */
Index tasks in the order of heaviest utilization to lightest;
for $i := 1$ to $N$ do
    Select $p$ such that $\sigma_p$ is minimal;
    if $U_i > 1 - \sigma_p$ then
        break /* this task must be migrating */
    fi;
    $s_{i,p},\ \sigma_p,\ last := U_i,\ \sigma_p + U_i,\ i$
od;
/* assign migrating and low-utilization fixed tasks */
$p := 1$;
for $i := last + 1$ to $N$ do
    $remaining := U_i$
    repeat
        $s_{i,p} := min(remaining,\ (1 - \sigma_p))$;
        $\sigma_p,\ remaining := \sigma_p + s_{i,p},\ remaining - s_{i,p}$;
        if $\sigma_p = 1$ then $p := p + 1$ fi
    until $remaining = 0$
od
```

Figure 4: EDF-os assignment phase.

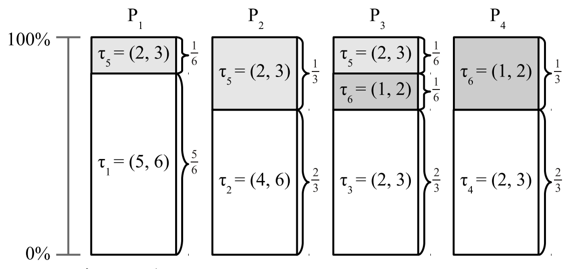

Figure 5: EDF-os task assignment for Ex. 1.

utilization 1.0, where each Pfair task corresponds to execution on a processor P_p and has utilization $f_{i,p}$, as before. If, in a uniprocessor schedule of these n Pfair tasks, the p^{th} task is allocated time slot t, then the t^{th} job of the migrating task is assigned to processor P_p. Because Prop. 1 is based solely on the guaranteed behavior of any Pfair scheduler, it holds for EDF-os with this generalized assignment policy.

Ex. 1 (revisited). We now discuss how EDF-os would schedule the task system from Ex. 1 in Sec. 2. For convenience, we list here the tasks in decreasing utilization order: $\tau = \{(5, 6), (4, 6), (2, 3), (2, 3), (2, 3), (1, 2)\}$. In Figs. 5–6, we also re-index the tasks to match this new ordering.

The task assignment EDF-os produces is shown in Fig. 5. Note that there are two migrating tasks, and one of them, $\tau_5 = (2, 3)$, executes on three processors, P_1, P_2, and P_3. Fig. 6 (with a key in Fig. 2) shows an example EDF-os schedule for this task system. In this case, due to the improved assignment scheme, only fixed tasks actually have deadline misses within the example schedule.

4 Tardiness Bounds

In this section, we derive tardiness bounds under EDF-os. We consider migrating and fixed tasks separately, in Secs. 4.1 and 4.2, respectively. For migrating tasks, we actually consider lateness bounds rather than tardiness bounds. Recall from Sec. 2 that if tardiness is positive, then lateness

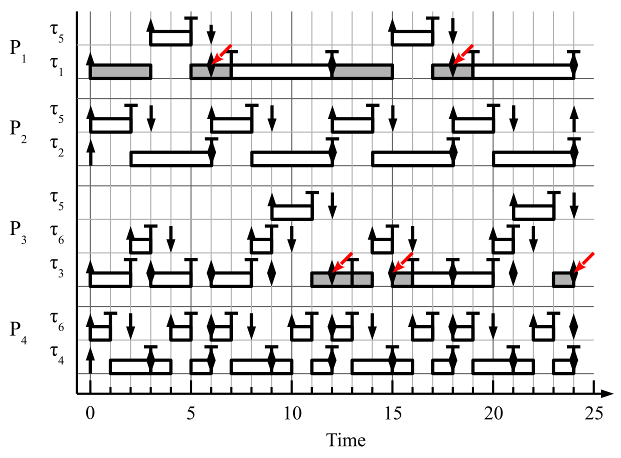

Figure 6: EDF-os schedule for Ex. 1. $f_{5,1} = \frac{1}{4}$, $f_{5,2} = \frac{1}{2}$ and $f_{5,3} = \frac{1}{4}$. $f_{6,3} = \frac{1}{3}$ and $f_{6,4} = \frac{2}{3}$.

is identical to tardiness, but lateness can be negative while tardiness cannot. Allowing the lateness bounds for migrating tasks to be negative can result in tighter tardiness bounds for fixed tasks. In the rest of this section, we assume that the task system τ being analyzed is feasible (refer to (2)). We denote the set of all fixed tasks on processor P_p as τ_p^f, and the sum of the shares of all fixed tasks on P_p as σ_p^f.

We begin by establishing several properties that follow from the assignment procedure in Fig. 4. Recall that, as discussed in Sec. 3, Prop. 1 holds for EDF-os.

Property 2. *For each migrating task τ_i, $U_i < 1$.*

This property follows from the worst-fit decreasing heuristic used by our assignment procedure. Because τ is feasible, if $U_i < 1$ fails to hold, then $U_i = 1$ holds. Moreover, $i \le M$, for otherwise, total utilization would exceed M. These facts imply that τ_i would have been assigned as a fixed task to a dedicated processor.

Property 3. *There are no more than two migrating tasks that assign jobs to processor P_p. If there are two migrating tasks that assign jobs to P_p, then P_p is the first processor for exactly one of them.*

It can be shown by induction that when our assignment procedure first considers a migrating task τ_i, there can be at most one migrating task already assigned to the currently considered processor (which will be τ_i's first processor). From this, Prop. 3 follows.

Property 4. *For processor P_p with one or more migrating tasks τ_i (and possibly τ_k) that have shares $s_{i,p}$ (and $s_{k,p}$), $\sigma_p^f + s_{i,p} + s_{k,p} \le 1$.*

Our assignment procedure does not allow σ_p to exceed 1.0 (i.e., P_p cannot be over-allocated).

Property 5. *If processor P_p contains migrating tasks τ_i and τ_k and P_p is the first processor of τ_k, then $s_{i,p} + U_k < 1$.*

Because tasks are assigned in decreasing-utilization order, there must be a fixed task τ_f on P_p such that $U_f \ge U_k$. Therefore, by Prop. 4 and because $s_{k,p} > 0$, Prop. 5 holds.

Property 6. *Out of any c consecutive jobs of some migrating task τ_i, the number of jobs released on P_p is at most $f_{i,p} \cdot c + 2$.*

By Prop. 1, if τ_i executes jobs on P_p, then out of its first n jobs, the number assigned to P_p is between $\lfloor f_{i,p} \cdot n \rfloor$ and $\lceil f_{i,p} \cdot n \rceil$. Thus, out of any c consecutive jobs of τ_i, where the index of the first such job is j, the number of jobs assigned to P_p is at most

$$\begin{aligned}
& \lceil f_{i,p} \cdot (j + c - 1) \rceil - \lfloor f_{i,p} \cdot (j-1) \rfloor \\
\le\ & \{\text{Since } \lceil x + y \rceil \le \lceil x \rceil + \lceil y \rceil\} \\
& \lceil f_{i,p} \cdot (j-1) \rceil + \lceil f_{i,p} \cdot c \rceil - \lfloor f_{i,p} \cdot (j-1) \rfloor \\
\le\ & \{\text{Since } \lceil x \rceil - \lfloor x \rfloor \le 1\} \\
& \lceil f_{i,p} \cdot c \rceil + 1 \\
<\ & \{\text{Since } \lceil x \rceil < x + 1\} \\
& f_{i,p} \cdot c + 2.
\end{aligned}$$

4.1 Lateness Bounds for Migrating Tasks

We now derive a lateness bound for migrating tasks. Since such tasks are statically prioritized over fixed ones, we need not consider fixed tasks in this derivation. Thus, all referenced tasks in this subsection are assumed to be migrating.

First, we provide a bound on the work from a migrating task that competes with an arbitrary task. This result will be used both here and in the next subsection.

Lemma 1. *Consider a migrating task τ_i that releases jobs on processor P_p. Let $t_0 \ge 0$ and $t_c > t_0$. If no job of τ_i has lateness exceeding Δ_i (which may be negative), then the demand from τ_i in the interval $[t_0, t_c)$ on P_p is less than*

$$(s_{i,p})(t_c - t_0) + (s_{i,p})(\Delta_i + 2T_i) + 2C_i.$$

Proof. Since we assume that the maximum lateness of τ_i is at most Δ_i, we know that any job released by τ_i will take no more than $T_i + \Delta_i$ time units to complete, so jobs of τ_i released before $t_0 - (\Delta_i + T_i)$ cannot create demand in $[t_0, t_c)$. Thus, competing demand for execution from jobs of τ_i in the interval $[t_0, t_c)$ comes from jobs of τ_i released in $[t_0 - \Delta_i - T_i, t_c)$. Since the minimum inter-release time between jobs of τ_i is T_i, there are at most $\left\lceil \frac{t_c - (t_0 - \Delta_i - T_i)}{T_i} \right\rceil$ such jobs released in this interval. Since τ_i is a migrating task, the number of jobs executed on P_p out of any number of consecutive jobs of τ_i is limited by Prop. 6. Thus, the demand from τ_i in the interval $[t_0, t_c)$ on P_p is at most

$$\begin{aligned}
& \left(f_{i,p} \cdot \left\lceil \frac{t_c - (t_0 - \Delta_i - T_i)}{T_i} \right\rceil + 2 \right) C_i \\
<\ & \{\text{Since } \lceil x \rceil < x + 1\} \\
& \left(f_{i,p} \cdot \left(\frac{t_c - (t_0 - \Delta_i - T_i)}{T_i} + 1 \right) + 2 \right) C_i \\
\le\ & \{\text{Rewriting}\} \\
& \left(f_{i,p} \cdot \left(\frac{t_c - t_0 + \Delta_i + 2T_i}{T_i} \right) + 2 \right) C_i
\end{aligned}$$

$$= \{\text{By (1)}\}$$
$$(s_{i,p})(t_c - t_0) + (s_{i,p})(\Delta_i + 2T_i) + 2C_i. \qquad \square$$

We now show that we can upper-bound the lateness of a migrating task τ_ℓ by using a reduction argument that considers an *alternate job allocation in which all of its jobs execute on its first processor, P_p*. (For ease of understanding, we use the indices "ℓ" and "h" in the rest of this subsection to reflect *lower* and *higher* static priorities, respectively.) Note that Prop. 5 ensures that, when ignoring fixed tasks (as we do in this subsection), P_p has sufficient capacity to accommodate any jobs of τ_ℓ we may move to it from other processors. This is because there must exist a fixed task on P_p with utilization at least that of τ_ℓ. (Our usage of a worst-fit decreasing assignment strategy is crucially exploited here.)

Lemma 2. *If every job of migrating task τ_ℓ that executes on a non-first processor of τ_ℓ is moved to its first processor P_p, no job of τ_ℓ will complete earlier. Also, if another migrating task τ_h executes on P_p, such moves do not affect it.*

Proof. If τ_ℓ shares P_p with another migrating task τ_h, then by the prioritization rules of EDF-os, τ_h is not impacted by moving jobs of τ_ℓ to P_p, since τ_h has higher priority than τ_ℓ (we are not changing the static prioritization of these tasks).

We now show that moving a single job $\tau_{\ell,k}$ of τ_ℓ to P_p cannot lessen the completion time of any job of τ_ℓ. By inducting over all such moves, the lemma follows.

Because job $\tau_{\ell,k}$ is being moved, it was originally executing on a non-first processor of τ_ℓ. Hence, $\tau_{\ell,k}$ was of highest priority on that processor and executed immediately to completion as soon as it was eligible (i.e., by the later of its release time and the completion time of its predecessor $\tau_{\ell,k-1}$, if any). After the move, its execution may be delayed by jobs of τ_h, which have higher priority than those of τ_ℓ on P_p. Thus, after the move, $\tau_{\ell,k}$ cannot complete earlier, and may complete later. If it completes later, then this cannot cause subsequent jobs of τ_ℓ to complete earlier (earlier jobs of τ_ℓ are clearly not impacted). $\square$

Thm. 1 below provides lateness bounds for migrating tasks. If a migrating task τ_ℓ shares its first processor with another migrating task τ_h, then the bound for τ_ℓ depends on that of τ_h. Such bounds can be computed inductively, with the following lemma providing the base case.

Lemma 3. *The migrating task τ_h with the lowest-indexed first processor P_p does not share P_p with another migrating task.*

Proof. By the assignment procedure of EDF-os, no migrating task other than τ_h executes on P_p. $\square$

Theorem 1. *Let P_p be the first processor of τ_ℓ. If τ_ℓ is not the only migrating task that executes on P_p, then let τ_h denote the unique (by Prop. 3) other migrating task that does so, and let Δ_h denote an upper bound on its lateness. Then, τ_ℓ has lateness no larger than*

$$\Delta_\ell \triangleq \begin{cases} \frac{(s_{h,p})(\Delta_h + 2T_h) + 2C_h + C_\ell}{1 - s_{h,p}} - T_\ell & \textit{if } \tau_h \textit{ exists,} \\ C_\ell - T_\ell & \textit{otherwise.} \end{cases} \quad (3)$$

Proof. By Lem. 2, we can establish the desired lateness bound by assuming that all jobs of τ_ℓ run on P_p. We make this assumption in the remainder of the proof.

If τ_ℓ is the only migrating task on P_p, then its jobs will be of highest priority on P_p. Thus, by Prop. 2 and Lem. 2, every job of τ_ℓ will have a response time of at most C_ℓ, and therefore a lateness of at most $C_\ell - T_\ell$.

In the rest of the proof, we assume that τ_ℓ shares P_p with another migrating task. By Prop. 3, there is a unique such task τ_h, as stated in the theorem. By the prioritization rules used by EDF-os, τ_h has higher priority than τ_ℓ.

Consider job $\tau_{\ell,j}$ with release time $r_{\ell,j}$ and deadline $d_{\ell,j}$. For purposes of contradiction, assume that $\tau_{\ell,j}$'s lateness exceeds Δ_ℓ. According to the prioritization rules used by EDF-os, $\tau_{\ell,j}$'s execution may be impacted only by jobs from τ_h and by jobs from τ_ℓ with deadlines before $d_{\ell,j}$. We now upper bound the processor demand impacting $\tau_{\ell,j}$ by considering a certain time interval, as defined next.

Interval $[t_0, t_c)$. Let t_0 be the latest point in time at or before $r_{\ell,j}$ such that no jobs of τ_h or τ_ℓ released on P_p before t_0 are pending; a released job is *pending* if it has not yet completed execution. (t_0 is well-defined because the stated condition holds at time 0.) Define $t_c \triangleq d_{\ell,j} + \Delta_\ell$. The assumption we seek to contradict is that $\tau_{\ell,j}$ does not complete by t_c. Since $\tau_{\ell,j}$ fails to complete by t_c, there are more than $t_c - t_0$ units of demand in the interval $[t_0, t_c)$ for the execution of jobs on P_p with priority at least that of $\tau_{\ell,j}$.

Demand from τ_h. By Lem. 1, the competing demand in $[t_0, t_c)$ due to τ_h on P_p is at most

$$(s_{h,p})(t_c - t_0) + (s_{h,p})(\Delta_h + 2T_h) + 2C_h. \quad (4)$$

Demand from τ_ℓ. Additional demand can come from jobs of τ_ℓ with deadlines earlier than $d_{\ell,j}$. By the definition of t_0, all such jobs are released in $[t_0, r_{\ell,j})$. Thus, there are at most $\left\lfloor \frac{(r_{\ell,j} - t_0)}{T_\ell} \right\rfloor$ such jobs. Including job $\tau_{\ell,j}$ itself, there are at most $\left\lfloor \frac{(r_{\ell,j} - t_0)}{T_\ell} \right\rfloor + 1$ jobs of τ_ℓ released in $[t_0, t_c)$ with deadlines at most $d_{\ell,j}$. The total demand due to such jobs is $\left(\left\lfloor \frac{(r_{\ell,j} - t_0)}{T_\ell} \right\rfloor + 1 \right) C_\ell$, which by the definition of U_ℓ is at most

$$U_\ell(r_{\ell,j} - t_0) + C_\ell. \quad (5)$$

Total demand. For notational convenience, let

$$K \triangleq (s_{h,p})(\Delta_h + 2T_h) + 2C_h + C_\ell. \quad (6)$$

Then, by (4) and (5), the total demand on P_p due to jobs of equal or higher priority than $\tau_{\ell,j}$ in $[t_0, t_c)$ is at most

$$K + (t_c - t_0)s_{h,p} + (r_{\ell,j} - t_0)U_\ell. \quad (7)$$

Because $\tau_{\ell,j}$ completed after time t_c (by assumption), the considered demand exceeds the length of $[t_0, t_c)$, so

$$(t_c - t_0) < \{\text{By (7)}\}$$
$$K + (t_c - t_0)s_{h,p} + (r_{\ell,j} - t_0)U_\ell$$

$$\begin{aligned}
&= \{\text{Rearranging}\}\\
&\quad K + (t_c - r_{\ell,j})s_{h,p} + (r_{\ell,j} - t_0)(s_{h,p} + U_\ell)\\
&< \{\text{By Prop. 5}\}\\
&\quad K + (t_c - r_{\ell,j})s_{h,p} + (r_{\ell,j} - t_0). \qquad (8)
\end{aligned}$$

Subtracting $(r_{\ell,j} - t_0)$ from both sides of (8) gives $(t_c - r_{\ell,j}) < K + (t_c - r_{\ell,j})s_{h,p}$, which implies

$$K > (t_c - r_{\ell,j})(1 - s_{h,p}). \qquad (9)$$

By Prop. 2, $U_h < 1$, and hence $s_{h,p} < 1$. Thus, by (9),

$$\begin{aligned}
(t_c - r_{\ell,j}) &< \{\text{since } 1 - s_{h,p} \text{ is positive}\}\\
&\quad \frac{K}{1 - s_{h,p}}\\
&= \{\text{By (6)}\}\\
&\quad \frac{(s_{h,p})(\Delta_h + 2T_h) + 2C_h + C_\ell}{1 - s_{h,p}}\\
&= \{\text{By (3)}\}\\
&\quad \Delta_\ell + T_\ell.
\end{aligned}$$

Because $r_{\ell,j} = d_{\ell,j} - T_\ell$, this implies $t_c - d_{\ell,j} < \Delta_\ell$, which contradicts the definition of t_c and thus violates our assumption that $\tau_{\ell,j}$ completes after time $d_{\ell,j} + \Delta_\ell$. □

4.2 Tardiness Bounds for Fixed Tasks

Although we provided bounds on lateness in Sec. 4.1, in this subsection we instead provide bounds on tardiness, because it is not possible for the bounds in this subsection to be negative. If no migrating tasks execute on a given processor, then the fixed tasks on that processor have zero tardiness, by the optimality of EDF on one processor. The following theorem establishes tardiness bounds for fixed tasks that must execute together with migrating tasks.

Theorem 2. *Suppose that at least one migrating task executes on processor P_p and let τ_i be a fixed task on P_p. If P_p has two migrating tasks (refer to Prop. 3), denote them as τ_h and τ_ℓ, where τ_h has higher priority; otherwise, denote its single migrating task as τ_h, and consider τ_ℓ to be a "null" task with $T_\ell = 1$, $s_{\ell,p} = 0$, and $C_\ell = 0$. Then, τ_i has a maximum tardiness of at most*

$$\Delta_i \triangleq \frac{(s_{h,p})(\Delta_h + 2T_h) + 2C_h + (s_{\ell,p})(\Delta_\ell + 2T_\ell) + 2C_\ell}{(1 - s_{h,p} - s_{\ell,p})}. \qquad (10)$$

Proof. The proof is similar to that of Thm. 1. We will upper bound demand over the following interval.

Interval $[t_0, t_c)$. For purposes of contradiction, suppose that there exists a job $\tau_{i,j}$ of τ_i that has tardiness exceeding Δ_i, i.e., $\tau_{i,j}$ has not completed by t_c, where $t_c \triangleq d_{i,j} + \Delta_i$. Define a job as a *competing* job if it is released on P_p and it is a job of τ_h or τ_ℓ, or a job of a fixed task that has a deadline at or before $d_{i,j}$. Let t_0 be the latest point in time at or before $r_{i,j}$ such that no competing jobs released before t_0 are pending. (t_0 is well-defined because the stated condition holds at time 0.) We now bound demand over $[t_0, t_c)$ due to competing jobs (including $\tau_{i,j}$ itself) by considering migrating and fixed tasks separately.

Demand from migrating tasks. By Lem. 1, demand over $[t_0, t_c)$ due to jobs of τ_h and τ_ℓ is at most

$$\begin{aligned}
&(s_{h,p})(t_c - t_0) + (s_{h,p})(\Delta_h + 2T_h) + 2C_h +\\
&(s_{\ell,p})(t_c - t_0) + (s_{\ell,p})(\Delta_\ell + 2T_\ell) + 2C_\ell. \qquad (11)
\end{aligned}$$

Demand from fixed tasks. A fixed task τ_k can release at most $\left\lfloor \frac{d_{i,j} - t_0}{T_k} \right\rfloor$ competing jobs within $[t_0, t_c)$. Thus, demand from all competing jobs of fixed tasks is at most

$$\sum_{\tau_k \in \tau_p^f} \left\lfloor \frac{d_{i,j} - t_0}{T_k} \right\rfloor C_k \leq (d_{i,j} - t_0) \sum_{\tau_k \in \tau_p^f} \frac{C_k}{T_k}. \qquad (12)$$

By the definition of σ_p^f, the bound in (12) can be written as

$$\begin{aligned}
(d_{i,j} - t_0)(\sigma_p^f) &\leq \{\text{By Prop. 4}\}\\
&\quad (d_{i,j} - t_0)(1 - s_{h,p} - s_{\ell,p}). \qquad (13)
\end{aligned}$$

Total demand. For notational convenience, let

$$K \triangleq (s_{h,p})(\Delta_h + 2T_h) + 2C_h + (s_{\ell,p})(\Delta_\ell + 2T_\ell) + 2C_\ell. \qquad (14)$$

Then, by (11) and (13), total competing demand is at most

$$\begin{aligned}
&K + s_{h,p}(t_c - t_0) + s_{\ell,p}(t_c - t_0) +\\
&(d_{i,j} - t_0)(1 - s_{h,p} - s_{\ell,p}). \qquad (15)
\end{aligned}$$

Because $\tau_{i,j}$ completed after time t_c (by assumption), the considered demand exceeds the length of the interval $[t_0, t_c)$, so

$$\begin{aligned}
(t_c - t_0) &< \{\text{By (15)}\}\\
&\quad K + s_{h,p}(t_c - t_0) + s_{\ell,p}(t_c - t_0) +\\
&\quad (d_{i,j} - t_0) - (d_{i,j} - t_0)(s_{h,p} + s_{\ell,p})\\
&= \{\text{Rearranging}\}\\
&\quad K + (s_{h,p} + s_{\ell,p})(t_c - t_0) +\\
&\quad (d_{i,j} - t_0) - (d_{i,j} - t_0)(s_{h,p} + s_{\ell,p}). \qquad (16)
\end{aligned}$$

Subtracting $(d_{i,j} - t_0)$ from both sides of (16), we have $(t_c - d_{i,j}) < K + (s_{h,p} + s_{\ell,p})(t_c - t_0) - (d_{i,j} - t_0)(s_{h,p} + s_{\ell,p}) = K + (s_{h,p} + s_{\ell,p})(t_c - d_{i,j})$. This implies

$$K > (t_c - d_{i,j})(1 - s_{h,p} - s_{\ell,p}). \qquad (17)$$

By Prop. 4 and because at least one fixed task τ_i is assigned to P_p, we have $(1 - s_{h,p} - s_{\ell,p}) > 0$. Thus, by (17),

$$
\begin{aligned}
t_c - d_{i,j} &< \frac{K}{(1 - s_{h,p} - s_{\ell,p})} \\
&= \{\text{By (10) and (14)}\} \\
&\quad \Delta_i.
\end{aligned}
$$

This contradicts our definition of $t_c = d_{i,j} + \Delta_i$, so it cannot be the case that $\tau_{i,j}$ has more than Δ_i units of tardiness. $\square$

In an online appendix [4], we discuss some possible improvements and extensions to EDF-os.

5 Experimental Comparison

Several scheduling algorithms have been previously evaluated for use in SRT systems. Bastoni et al. [10] compared several semi-partitioned algorithms, including EDF-fm and also EDF-WM [26], although the latter was originally designed for HRT systems. In that study, EDF-WM was shown to be effective for SRT systems due to its low overheads. Although not a semi-partitioned algorithm, the global algorithm G-FL has been proposed by Erickson et al. [19] as a promising scheduler for SRT systems. G-FL has provably better tardiness bounds than the better known G-EDF algorithm. The variant C-FL [20], which partitions tasks onto clusters of processors and runs G-FL within each cluster, is often preferable in the presence of overheads.

We conducted overhead-aware experiments in which each of EDF-os, EDF-fm, EDF-WM, and C-FL were compared on the basis of schedulability and the tardiness bounds they ensure. In order to determine the effect of overheads, we implemented EDF-os in LITMUS$^{\mathrm{RT}}$ [1] and measured the same scheduler-specific overheads considered by Bastoni et al. [10]. Our modifications to LITMUS$^{\mathrm{RT}}$ are available at [1]. We used an Intel Xeon L7455 system, which has 24 cores on four physical sockets. The cores in each socket share an L3 cache, and pairs of cores share an L2 cache. Overheads on the *same* machine were available for EDF-fm and EDF-WM from the study in [10] and for C-FL from the study in [20]. Cache-related preemption and migration delays were also measured on this machine in a prior study [8]. All of those prior measurements were reused in the study here. We used these overheads in an overhead-aware schedulability study involving randomly generated task sets following the methodology in [10]. The code used for schedulability tests is included with the online appendix [4]. These experiments were conducted in order to augment the study of Bastoni et al. by including EDF-os (and also C-FL). It is beyond the scope of this paper to conduct a thorough evaluation of all relevant semi-partitioned algorithms that have been proposed. (We note that the experiments herein are not merely simulations; each tested algorithm requires an *actual kernel implementation* so that overheads can be measured.)

In our experiments, we randomly generated implicit-deadline task sets, inflated the task system parameters to account for average-case[2] observed overheads, and computed the resulting *schedulability*—defined as the fraction of generated systems for which bounded tardiness can be guaranteed—and *maximum tardiness bounds* under each tested algorithm. Task utilizations were generated using uniform, bimodal, and exponential distributions as in [10]. For *uniform* distributions, we considered a *light* distribution where values were drawn from $[0.001, 0.1]$, a *medium* distribution where values were drawn from $[0.1, 0.4]$, and a *heavy* distribution where values were drawn from $[0.5, 0.9]$. For *bimodal* distributions, we drew values uniformly in the range of either $[0.001, 0.05]$ or $[0.5, 0.9]$ with respective probabilities of either $\frac{8}{9}$ and $\frac{1}{9}$, $\frac{6}{9}$ and $\frac{3}{9}$, or $\frac{4}{9}$ and $\frac{5}{9}$, for *light*, *medium*, and *heavy* distributions, respectively. For *exponential* distributions, we used a respective mean of 0.1, 0.25, and 0.5 for *light*, *medium*, and *heavy* distributions, respectively, and discarded any values that exceeded one. We generated periods uniformly from either a *short* (3 ms to 33 ms), *moderate* (10 ms to 100 ms), or *long* (50 ms to 250 ms) distribution.

We also considered utilization caps in the set $\{1, 1.25, 1.5, \ldots, 24\}$, and working set sizes (WSSs) from 16 KB to 3072 KB. WSSs from $[0, 256)$ KB were considered in increments of 16 KB, from $[256, 1024)$ KB in increments of 64 KB, and from $[1024, 3072]$ KB in increments of 256KB.

We generated 100 task sets for each combination of period distribution, utilization distribution, utilization cap, and WSS. When generating each task set, we added tasks until the total utilization exceeded the utilization cap, and then removed the last task. We considered several variants of the four tested schedulers, EDF-os, EDF-fm, EDF-WM, and C-FL, yielding ten possibilities in total. We used clustering based on L3 cache boundaries for C-FL, resulting in clusters of size six; this choice was made because C-FL has overheads similar to clustered EDF (C-EDF), and [10] used C-EDF with L3 cache boundaries as its standard of comparison. (C-FL, which was developed after the publication of [10], has better tardiness bounds than C-EDF.) For each tested scheduler, we considered cache-related preemption and migration delays both for an *idle* system and a system under *load*; considering both possibilities allows conclusions to be drawn for systems with light and heavy cache contention, respectively. Finally, because EDF-WM was designed as a HRT scheduler, it may not behave correctly if overheads cause jobs to miss deadlines (specifically, such misses may cause a task to run in parallel with itself). Therefore, for EDF-WM, we also considered behavior in the presence of worst-case observed overheads (denoted with *wc*), as doing so is probably necessary in practice.

Schedulability depends on both WSS and the assumed utilization cap. To avoid having to create three-dimensional graphs, we use the metric of *weighted schedulability* from [9]. Let $S(U, W) \in [0, 1]$ denote the schedulability of an algorithm (after accounting for overheads) with utilization cap (before overheads) U and WSS W, and let Q denote the set of considered utilization caps. *Weighted schedulability*,

[2]In prior studies, e.g., [9, 10], average-case overheads were considered when evaluating SRT schedulers, and worst-case overheads when evaluating HRT schedulers.

$S(W)$, is defined as $S(W) = \frac{\sum_{U \in Q} U \cdot S(U,W)}{\sum_{U \in Q} U}$. A conventional utilization-based schedulability plot for a fixed WSS collapses to a single point in a weight schedulability graph that gives the total area under the conventional plot [9].

Due to space constraints, we present only a subset of our results here—other results can be found in an online appendix [4]. A typical result for weighted schedulability is depicted in Fig. 7, which shows weighted schedulability, with respect to WSS, for each algorithm under uniform medium utilizations and uniform moderate periods.[3] Because all utilizations considered are no greater than 0.5, EDF-fm has 100% schedulability before accounting for overheads. EDF-os and EDF-fm have similar overheads, so they are nearly indistinguishable from a schedulability perspective, but both have higher schedulability than EDF-WM or C-FL. Fig. 8 has the same axes as Fig. 7, but depicts a uniform heavy utilization distribution rather than a uniform medium distribution. Because the utilization constraint for EDF-fm is no longer guaranteed to be satisfied, EDF-fm schedules very few task systems in this case. However, EDF-os continues to exhibit the best weighted schedulability of any considered algorithm. Overall, EDF-os usually provided the best weighted schedulability of any algorithm, although EDF-fm and EDF-WM sometimes provided small advantages for task systems with light utilizations.

Tardiness bounds are depicted with respect to utilization cap (for a fixed WSS of 128 KB) in Fig. 9. For small utilization caps, C-FL can guarantee negative lateness, which leads to a tardiness bound of zero. Usually, fewer tasks are migratory under EDF-os than under EDF-fm; as a result, tardiness was usually drastically lower under EDF-os than under EDF-fm, often very close to zero. Therefore, even for task systems where EDF-fm and EDF-os yielded comparable schedulability, EDF-os was superior. Overall, C-FL typically provided tardiness bounds between those of EDF-fm and EDF-os, while EDF-WM provided zero tardiness (as it is a HRT scheduler), at the cost of the inability to schedule many task sets. C-FL sometimes provided smaller tardiness bounds than EDF-os for some task systems with small WSSs where migration overheads are relatively small, but typically only EDF-WM yielded smaller tardiness bounds than EDF-os.

Because EDF-WM is not boundary-limited, it might not always interact well with synchronization protocols in the presence of critical sections [15]. Therefore, in addition to its typically better schedulability, EDF-os provides a significant practical advantage over EDF-WM when synchronization is needed. Moreover, as noted previously, the behavior of EDF-WM is not well-defined if systems are provisioned assuming deadline misses are tolerable. Compared to C-FL, EDF-os provides better schedulability and typically

[3]Under the experimental process followed in [10], "load" and "idle" curves should really be displayed separately because they cannot be directly compared. For example, for certain WSSs, it is possible that under load, cache-related overheads are lower, leading to improved schedulability compared to the idle case, because no affinity with the cache is established. We have combined these curves due to space constraints.

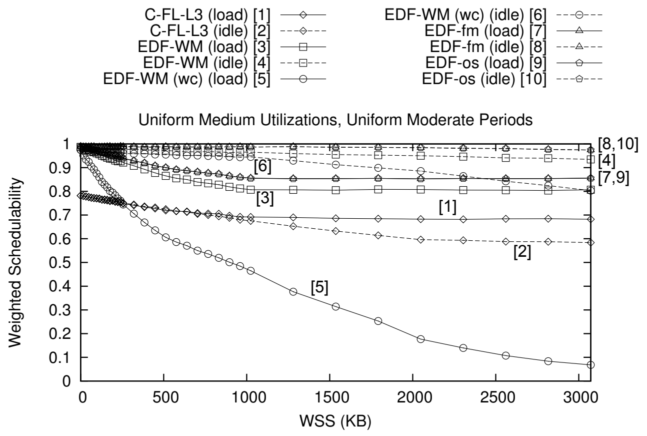

Figure 7: Weighted schedulability for task systems with uniform medium utilizations and uniform moderate periods.

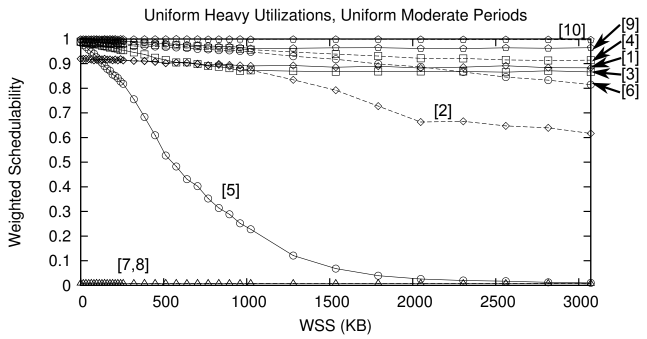

Figure 8: Weighted schedulability for task systems with uniform heavy utilizations and uniform moderate periods.

provides lower tardiness bounds, and compared to EDF-fm, EDF-os provides both better schedulability and lower tardiness bounds. Therefore, EDF-os represents a significant improvement to the state-of-the-art for SRT scheduling.

6 Conclusion

We have closed a long-standing open problem by presenting EDF-os, the first boundary-limited semi-partitioned scheduling algorithm that is optimal under the "bounded tardiness" definition of SRT correctness. We have also discussed (in an online appendix [4]) optimal variants of EDF-os in which implicit deadlines are not assumed and in which algorithms other than EDF are used as the secondary scheduler. EDF-os and its analysis improve upon prior work on EDF-fm by introducing two new key ideas: using some static prioritizations to make the execution of migrating tasks more predictable; and exploiting properties of worst-fit decreasing task assignments to enable a migrating task to be analyzed by "pretending" that all of its jobs execute on its first processor. In experiments that we conducted, EDF-os proved to be the best overall alternative from a schedulability perspective while providing very low tardiness bounds. Moreover, it has practical advantages over algorithms that are not boundary-limited.

The only other optimal boundary-limited scheduling al-

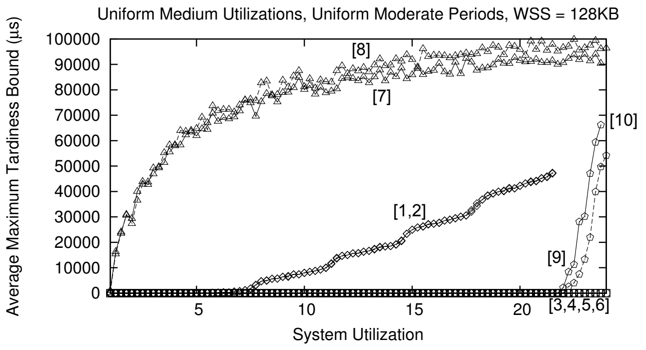

Figure 9: Tardiness bounds for schedulable systems with uniform medium utilizations, uniform moderate periods, and a WSS of 128 KB.

gorithms for SRT systems known to us are non-preemptive global EDF (NP-G-EDF) [17] and global FIFO (G-FIFO) [29] (which is also non-preemptive). For static systems, EDF-os is likely to be preferable in practice, because the tardiness bounds we have established are much lower than those known for NP-G-EDF and G-FIFO, and because semi-partitioned algorithms have lower runtime overheads than global ones [10]. On the other hand, for dynamic systems, where task timing parameters (such as execution budgets and periods) may change at runtime, NP-G-EDF is likely to be preferable, as EDF-based global scheduling tends to more amenable to runtime changes [14]. In contrast, the correctness of EDF-os relies crucially on how tasks are assigned to processors, and redefining such assignments on-the-fly does not seem easy.

As discussed in the online appendix [4], the conditions we present here for bounded tardiness are not sufficient if non-preemptive code regions exist. We would like to determine tight conditions that guarantee bounded tardiness in such circumstances. Furthermore, a job splitting technique as in [20] might be useful to reduce tardiness bounds, even after accounting for the increased overheads resulting from such a technique. For example, a task with $C_i = 300$ and $T_i = 1000$ could have each job split into ten subjobs, resulting in a task with $C_i = 30$ and $T_i = 100$. We would like to examine the effects of job splitting under EDF-os.

References

[1] LITMUS$^{\mathrm{RT}}$ home page. http://www.litmus-rt.org/.

[2] J. Anderson, V. Bud, and U. Devi. An EDF-based scheduling algorithm for multiprocessor soft real-time systems. In *17th ECRTS*, 2005.

[3] J. Anderson, V. Bud, and U. Devi. An EDF-based restricted-migration scheduling algorithm for multiprocessor soft real-time systems. *Real-Time Sys.*, 38(2):85–131, 2008.

[4] J. Anderson, J. Erickson, U. Devi, and B. Casses. Appendix to optimal semi-partitioned scheduling in soft real-time systems. `http://cs.unc.edu/~anderson/papers.html`, January 2014.

[5] B. Andersson, K. Bletsas, and S. Baruah. Scheduling arbitrary-deadline sporadic task systems on multiprocessors. In *29th RTSS*, 2008.

[6] B. Andersson and E. Tovar. Multiprocessor scheduling with few preemptions. In *12th RTCSA*, 2006.

[7] S. Baruah, N. Cohen, C.G. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, 1996.

[8] A. Bastoni, B. Brandenburg, and J. Anderson. Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. In *6th OSPERT*, 2010.

[9] A. Bastoni, B. Brandenburg, and J. Anderson. An empirical comparison of global, partitioned, and clustered multiprocessor EDF schedulers. In *21st RTSS*, 2010.

[10] A. Bastoni, B.B. Brandenburg, and J. Anderson. Is semi-partitioned scheduling practical? In *23rd ECRTS*, 2011.

[11] M. Bhatti, C. Belleudy, and M. Auguin. A semi-partitioned real-time scheduling approach for periodic task systems on multicore platforms. In *27th SAC*, 2012.

[12] K. Bletsas and B. Andersson. Notional processors: an approach for multiprocessor scheduling. In *15th RTAS*, 2009.

[13] K. Bletsas and B. Andersson. Preemption-light multiprocessor scheduling of sporadic tasks with high utilisation bound. *Real-Time Sys.*, 47(4):319–355, 2011.

[14] A. Block. *Multiprocessor Adaptive Real-Time Systems*. PhD thesis, University of North Carolina, Chapel Hill, NC, 2008.

[15] B. Brandenburg. *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*. PhD thesis, The University of North Carolina, Chapel Hill, NC, 2011.

[16] A. Burns, R. Davis, P. Wang, and F. Zhang. Partitioned EDF scheduling for multiprocessors using a C=D task splitting scheme. *Real-Time Sys.*, 48(1):3–33, 2012.

[17] U. Devi and J. Anderson. Tardiness bounds for global EDF scheduling on a multiprocessor. *Real-Time Sys.*, 38(2):133–189, 2008.

[18] F. Dorin, P. Yomsi, J. Goossens, and P. Richard. Semi-partitioned hard real-time scheduling with restricted migrations upon identical multiprocessor platforms. *Cornell University Library Archives*, arXiv:1006.2637 [cs.OS], 2010.

[19] J. Erickson and J. Anderson. Fair lateness scheduling: Reducing maximum lateness in g-edf-like scheduling. In *24th ECRTS*, 2012.

[20] J. Erickson and J. Anderson. Reducing tardiness under global scheduling by splitting jobs. In *25th ECRTS*, 2013.

[21] M. Fan and G. Quan. Harmonic semi-partitioned scheduling for fixed-priority real-time tasks on multi-core platform. In *Proc. of DATE*, 2012.

[22] J. Goossens, P. Richard, M. Lindström, I. Lupu, and F. Ridouard. Job partitioning strategies for multiprocessor scheduling of real-time periodic tasks with restricted migrations. In *20th RTNS*, 2012.

[23] N. Guan, M. Stigge, W. Yi, and G. Yu. Fixed-priority multiprocessor scheduling: Beyond Liu & Layland utilization bound. In *31st RTSS WiP*, 2010.

[24] N. Guan, M. Stigge, W. Yi, and G. Yu. Fixed-priority multiprocessor scheduling with Liu and Layland's utilization bound. In *16th RTAS*, 2010.

[25] S. Kato and N. Yamasaki. Real-time scheduling with task splitting on multiprocessors. In *13th RTCSA*, 2007.

[26] S. Kato and N. Yamasaki. Portioned EDF-based scheduling on multiprocessors. In *8th EMSOFT*, 2008.

[27] S. Kato and N. Yamasaki. Semi-partitioning technique for multiprocessor real-time scheduling. In *29th RTSS WiP*, 2008.

[28] S. Kato and N. Yamasaki. Semi-partitioned fixed-priority scheduling on multiprocessors. In *15th RTAS*, 2009.

[29] H. Leontyev and J. Anderson. Tardiness bounds for FIFO scheduling on multiprocessors. In *19th ECRTS*, 2007.

[30] H. Leontyev and J. Anderson. Generalized tardiness bounds for global multiprocessor scheduling. *Real-Time Sys.*, 44(1):26–71, 2010.

[31] A. Mills and J. Anderson. A multiprocessor server-based scheduler for soft real-time tasks with stochastic execution demand. In *17th RTCSA*, 2011.

[32] M. Shekhar, A. Sarkar, H. Ramaprasad, and F. Mueller. Semi-partitioned hard real-time scheduling under locked cache migration in multicore systems. In *24th ECRTS*, 2012.

[33] P. Sousa, P. Souto, E. Tovar, and K. Bletsas. The carousel-EDF scheduling algorithm for multiprocessor systems. In *19th RTCSA*, 2013.

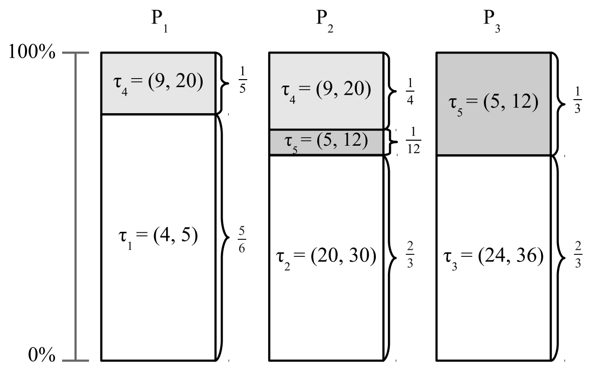

Figure 10: Counterexample to show that EDF-os is not optimal for non-preemptive task systems.

A Appendix: Additional Material

In this appendix, we present our experimental methodology in full and discuss several extensions to our work.

Potential Extensions to EDF-os

Non-preemptive sections. After designing EDF-os, we initially thought it retained its optimality if job execution is non-preemptive. However, this turns out not to be the case. In particular, with non-preemptivity, a job of a migrating task executing on a non-first processor for that task may be non-preemptively blocked when it is released. This blocking negates an important property exploited in our analysis, namely that such jobs execute immediately upon release. Here we give a counterexample consisting of five tasks executing on three processors where such non-preemptive blocking causes a migrating task to have unbounded tardiness.

Let $\tau = \{\tau_1 = (4, 5), \tau_2 = (20, 30), \tau_3 = (24, 36), \tau_4 = (9, 20), \tau_5 = (5, 12)\}$ and $M = 3$. Since $U(\tau) = 3.0$, τ is feasible on three processors. The assignment phase of EDF-os would assign the five tasks as shown in Fig. 10. In this example, tardiness can be unbounded for τ_5 if jobs are released as follows. Let the first job of τ_5 be released at time 1 and periodically once every 12 time units thereafter. Since $f_{5,3} = 4/5$ and $f_{5,2} = 1/5$, consider a job assignment in which the first four of every group of five jobs $5n + 1 \ldots 5n + 5$ (*i.e.*, jobs $5n + 1 \ldots 5n + 4$) of τ_5 are assigned to P_3 and the last job (job $5n + 5$) to P_2 for all $n \geq 0$. Let τ_3 (τ_2) release a job one time unit before the first of the four jobs (fifth job) of every five jobs of τ_5 assigned to P_3 (P_2) becomes eligible. Let τ_4's jobs assigned to P_2 be released at exactly the same time that a job of τ_5 assigned to P_2 become eligible. ($f_{4,1} = 4/9$ and $f_{4,2} = 5/9$, and hence, jobs 1, 3, 5, 7, and 8 of every group of nine jobs $9n + 1 \ldots 9n + 9$, $n \geq 0$, can be assigned to P_2, and the remaining jobs to P_1. Since $p_4 = 20$, it is sufficient if the separation between two consecutive jobs of τ_4 assigned to P_2 is 40 time units. With only every fifth job of τ_5 assigned to P_2, the eligibility times of two consecutive jobs of τ_5 assigned to P_2 is at least 60. Thus, τ_4's jobs can be released such that releases on P_2 coincide with the eligibility times of jobs of τ_5 assigned to P_2.) With such a job release pattern, the first of every group of four jobs of τ_5 assigned to P_3 is blocked by τ_3 for 23 time units after it becomes eligible. (The remaining three jobs are eligible when their predecessors complete executing and hence do not incur additional blocking.) Similarly, every fifth job is blocked for 19 time units due to τ_2 and waits for an additional 9 time units due to τ_4, for a total waiting time of 28 time units. Thus, 51 time units in every 60 time units within which every five jobs of τ_5 need to execute are spent waiting on other jobs. Hence, since the total execution requirement for five jobs is 25, tardiness for jobs in each group increases by 16 and grows unboundedly.

Refined assignment procedures. Our analysis suggests that, by refining EDF-os's assignment procedure, it may possible to obtain lower lateness/tardiness bounds. First, note that the inductive nature of the lateness bound calculation for migrating tasks may cause migrating tasks assigned to later processors to have higher bounds because lateness can cascade (though it will remain bounded). It may be possible to reduce such cascades by adjusting the assignment of migrating tasks, particularly on systems that are not fully utilized. Second, note that reducing the shares of migrating tasks executing on P_p reduces the bounds in (3) and (10). However, such a reduction would entail increasing the shares of these tasks on other processors, which could lead to lateness/tardiness bound increases on those processors. It may be possible to take such linkages among processors into account and obtain an assignment of tasks to processors that lessens the largest tardiness bound in the system. Finally, note that (3) includes $-T_\ell$ as part of τ_ℓ's lateness bound. By biasing the task assignment procedure to prefer larger periods for migrating tasks, it might be possible to lessen the lateness/tardiness bounds that result. We leave refinements to our assignment procedure motivated by these observations as future work.

Bounds with non-implicit deadlines. We have so far assumed that all job deadlines are implicit. However, if we maintain the prioritizations that EDF-os uses, then bounded tardiness can be easily ensured for systems with non-implicit deadlines, i.e., ones where each task τ_i has a specified relative D_i that may be less than, equal to, or greater than T_i. Maintaining the existing prioritizations for migrating tasks is straightforward, as these tasks are not scheduled by deadline (that are statically prioritized). For each job $\tau_{i,j}$ of a fixed task τ_i, we merely need to define a "scheduling deadline" equal to $r_{i,j} + T_i$ and prioritize such jobs on an EDF basis using scheduling deadlines instead of real ones. With this change, EDF-os will behave as before, but our analysis then bounds lateness/tardiness (for both migrating and fixed tasks) with respect to scheduling deadlines. However, such bounds can be easily corrected to be expressed with respect to real deadlines: if $D_i > T_i$, then simply subtract $D_i - T_i$ from the bound; if $D_i < T_i$, then simply add $T_i - D_i$ to the bound.

Window constrained second-level schedulers In defining EDF-os, we used EDF as a secondary scheduler for fixed tasks. (For migrating tasks, our prioritization rules and the sporadic task model fully characterize the behavior.)

Optimal variants of EDF-os can be constructed in which other algorithms are used as the secondary scheduler. All that we require is that a *window-constrained* [30] scheduler be used. Such a scheduler employs a per-task priority function $\chi_i(\tau_{i,j}, t)$ such that for some constants ϕ_i and ψ_i, $r_{i,j} - \phi_i \leq \chi_i(\tau_{i,j}, t) \leq d_{i,j} + \psi_i$ for each job $\tau_{i,j}$. The priority of job $\tau_{i,j}$ is at least that of $\tau_{i',j'}$ at time t if $\chi_i(\tau_{i,j}, t) \leq \chi_{i'}(\tau_{i',j'}, t)$ (priority functions can potentially change with time).

Our analysis can be modified to deal with this more general priority specification as follows. The bounds for migrating tasks continue to hold without modification; the proof of Thm. 1 is unchanged. By the definition of EDF-os priorities, all jobs of τ_h always have a higher priority than $\tau_{\ell,j}$, and by the sporadic task model, no job of τ_ℓ released after $r_{\ell,j}$ is eligible for execution before $\tau_{\ell,j}$ completes.

However, in our analysis of the tardiness of fixed tasks in the proof of Theorem 2, the number of competing jobs due to a fixed task τ_k is no longer $\left\lfloor \frac{d_{i,j}-t_0}{T_k} \right\rfloor$ but $\left\lfloor \frac{d_{i,j}+\psi_i+\phi_k-t_0}{T_k} \right\rfloor$. In effect, this change causes "$d_{i,j}$" to be replaced by "$d_{i,j}+\psi_i$" throughout the proof and K to be inflated by an additional $\sum_{\tau_k \in \tau_p^f} U_k \phi_k$. As a result, Δ_i must be increased by $\psi_i + \frac{\sum_{\tau_k \in \tau_p^f} U_k \phi_k}{1 - s_{h,p} - s_{\ell,p}}$. Tighter analysis may be possible if more is known about the prioritization function (as was the case with EDF).

B Appendix: Additional Graphs

In this appendix, we provide full graphs for our experiments. In order to keep the number of graphs manageable, when considering tardiness bounds we either fixed the WSS at 128 KB or fixed the utilization cap at 20. To facilitate grouping of figures for similar distributions, we begin the graphs on the next page.

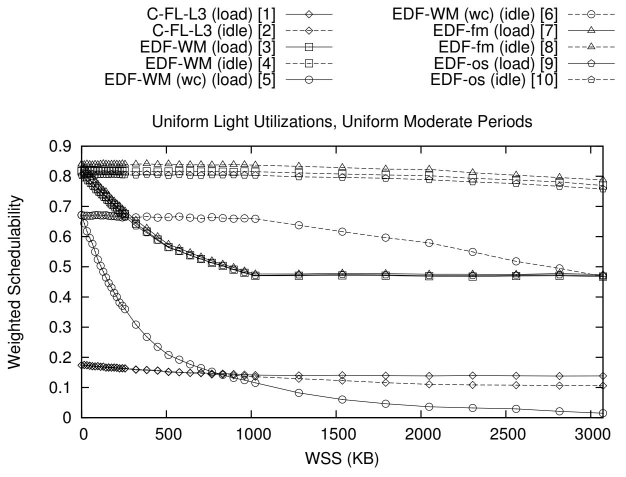

Figure 11: Uniform Light Utilizations, Uniform Moderate Periods

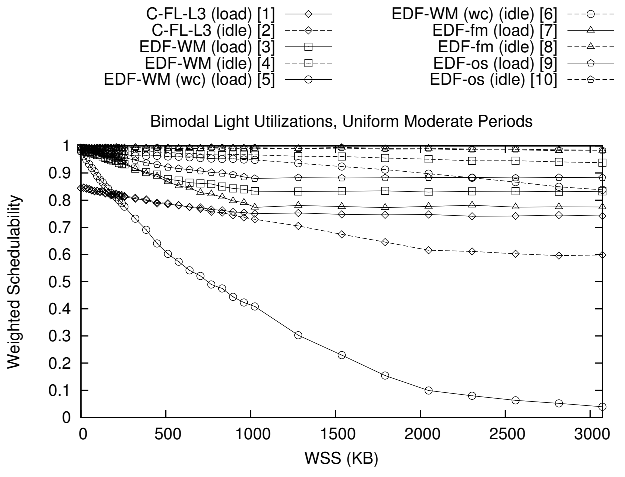

Figure 14: Bimodal Light Utilizations, Uniform Moderate Periods

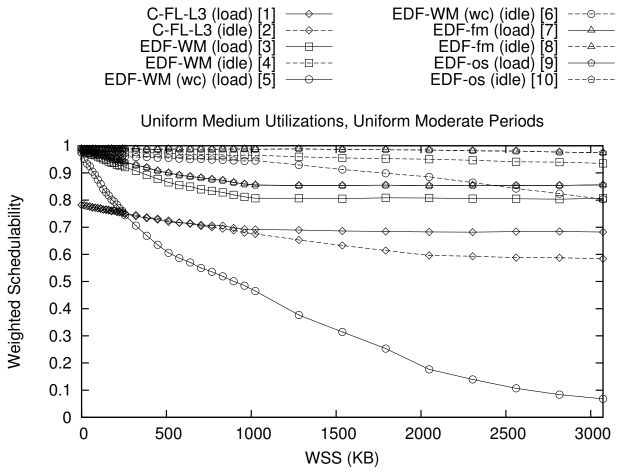

Figure 12: Uniform Medium Utilizations, Uniform Moderate Periods

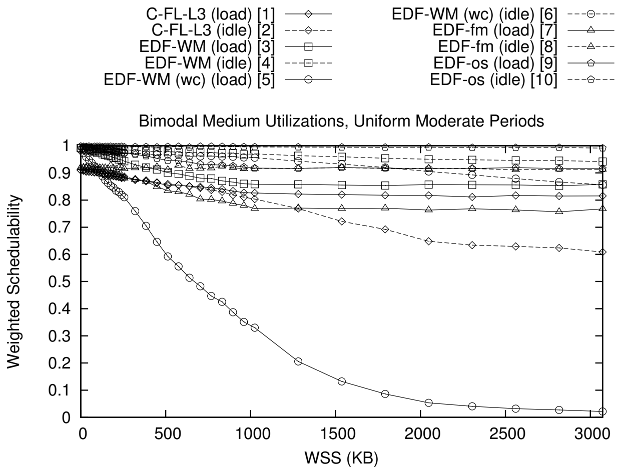

Figure 15: Bimodal Medium Utilizations, Uniform Moderate Periods

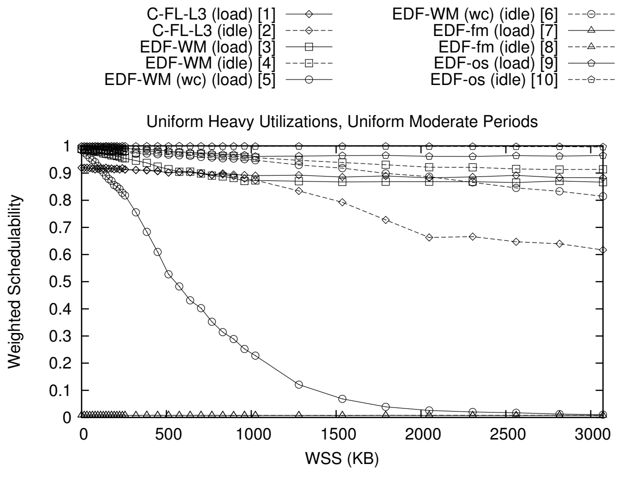

Figure 13: Uniform Heavy Utilizations, Uniform Moderate Periods

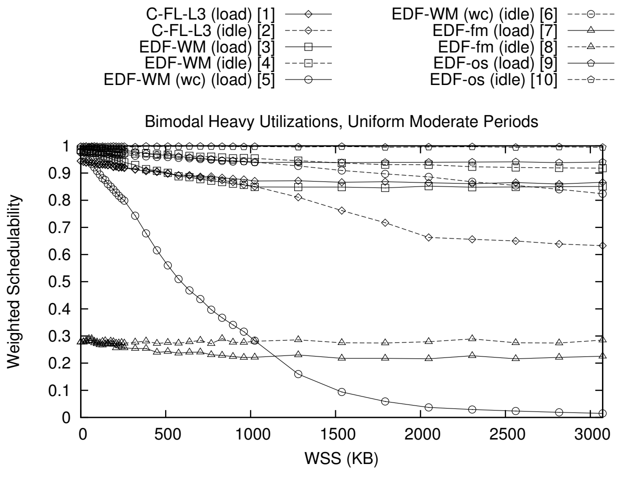

Figure 16: Bimodal Heavy Utilizations, Uniform Moderate Periods

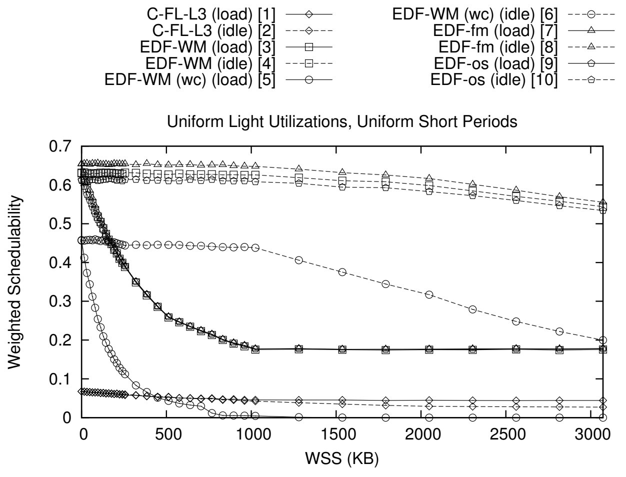

Figure 17: Uniform Light Utilizations, Uniform Short Periods

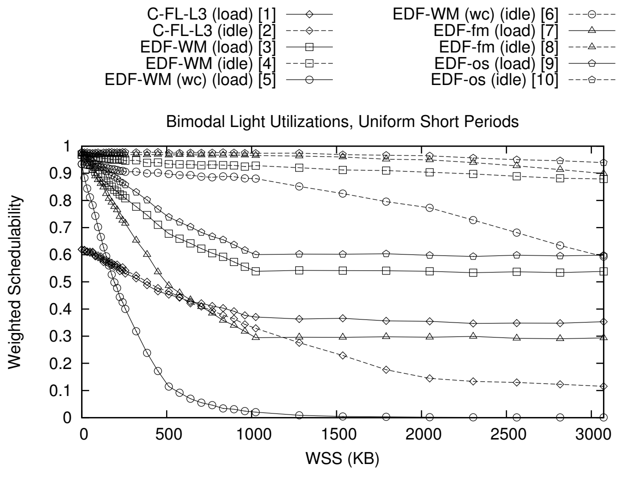

Figure 20: Bimodal Light Utilizations, Uniform Short Periods

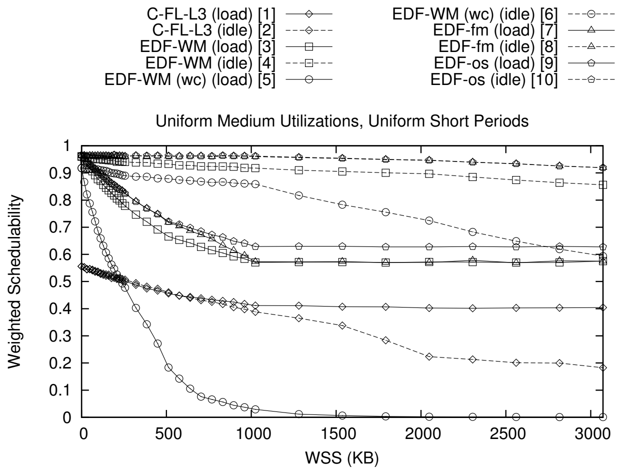

Figure 18: Uniform Medium Utilizations, Uniform Short Periods

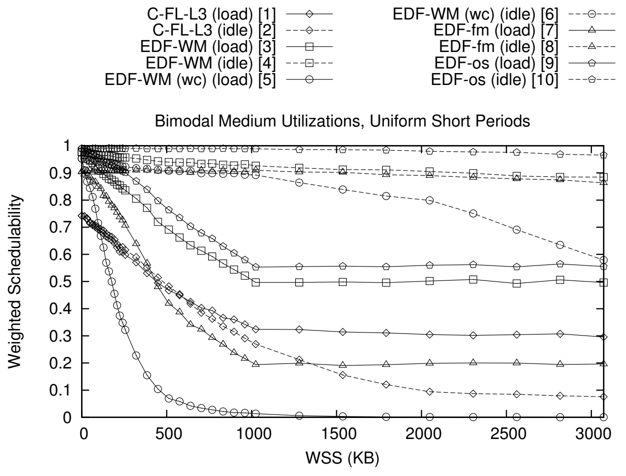

Figure 21: Bimodal Medium Utilizations, Uniform Short Periods

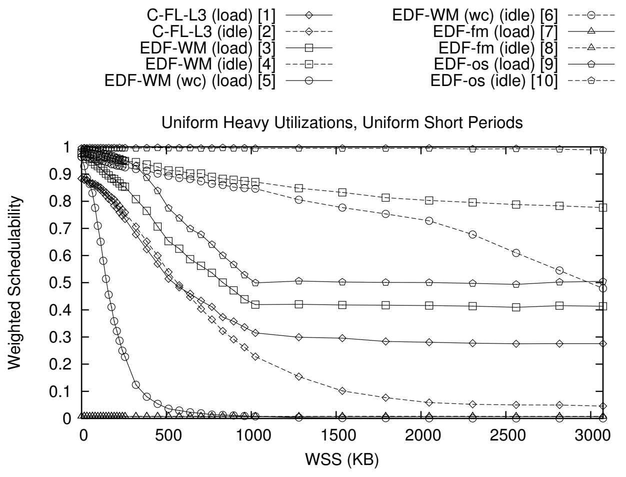

Figure 19: Uniform Heavy Utilizations, Uniform Short Periods

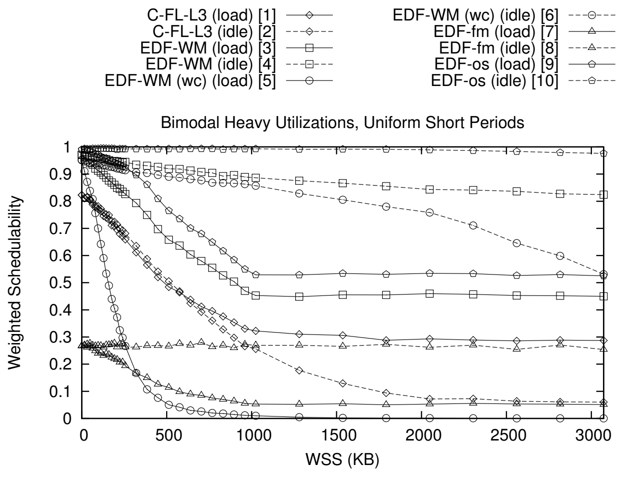

Figure 22: Bimodal Heavy Utilizations, Uniform Short Periods

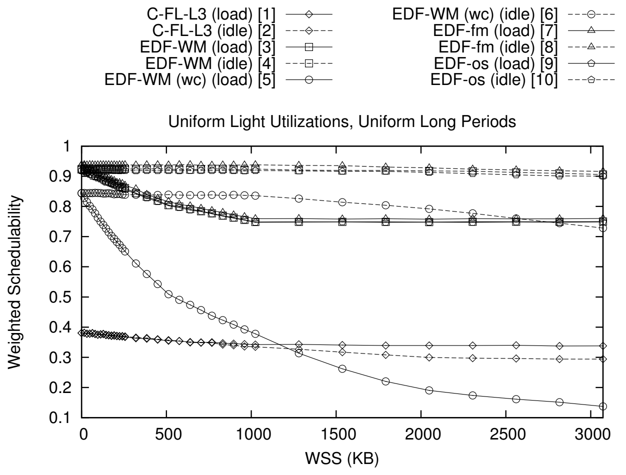

Figure 23: Uniform Light Utilizations, Uniform Long Periods

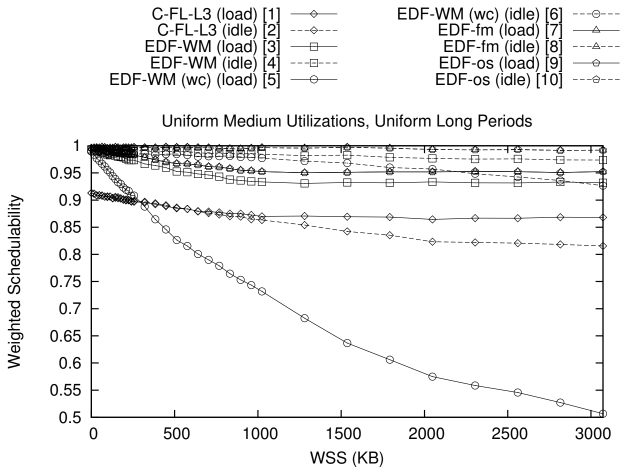

Figure 24: Uniform Medium Utilizations, Uniform Long Periods

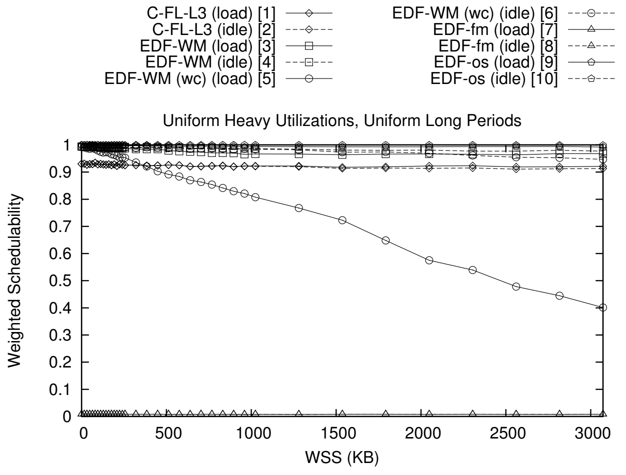

Figure 25: Uniform Heavy Utilizations, Uniform Long Periods

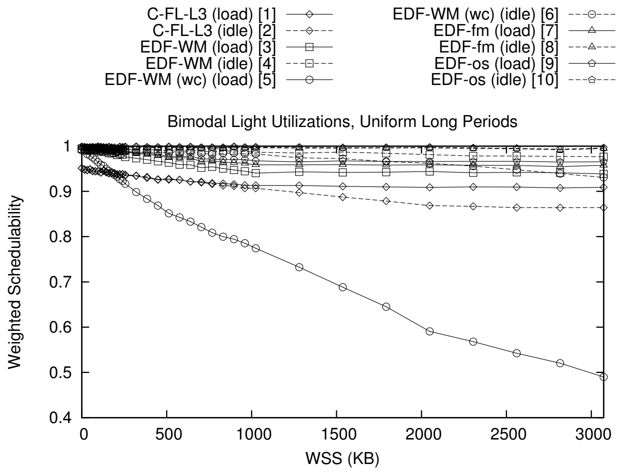

Figure 26: Bimodal Light Utilizations, Uniform Long Periods

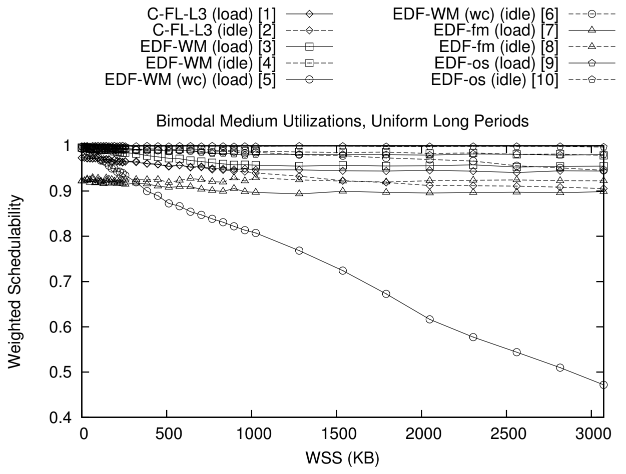

Figure 27: Bimodal Medium Utilizations, Uniform Long Periods

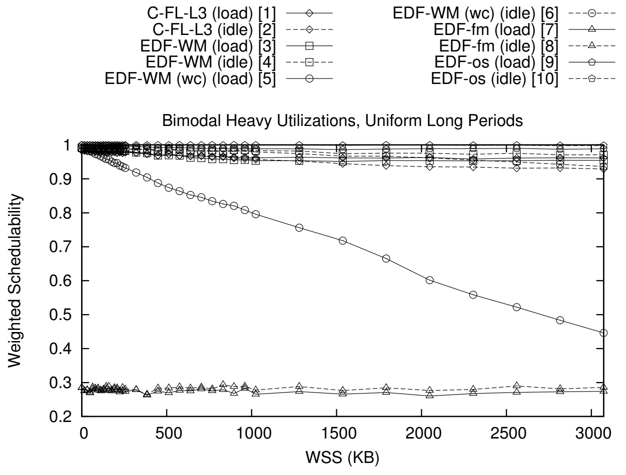

Figure 28: Bimodal Heavy Utilizations, Uniform Long Periods

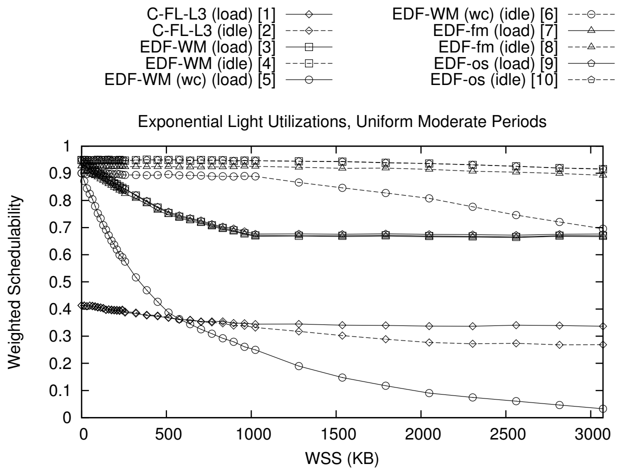

Figure 29: Exponential Light Utilizations, Uniform Moderate Periods

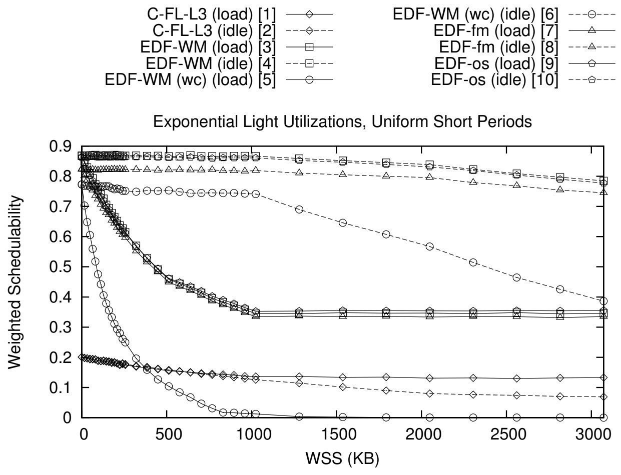

Figure 32: Exponential Light Utilizations, Uniform Short Periods

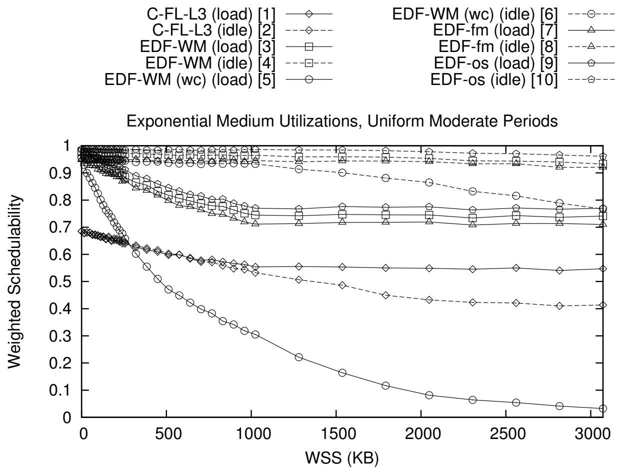

Figure 30: Exponential Medium Utilizations, Uniform Moderate Periods

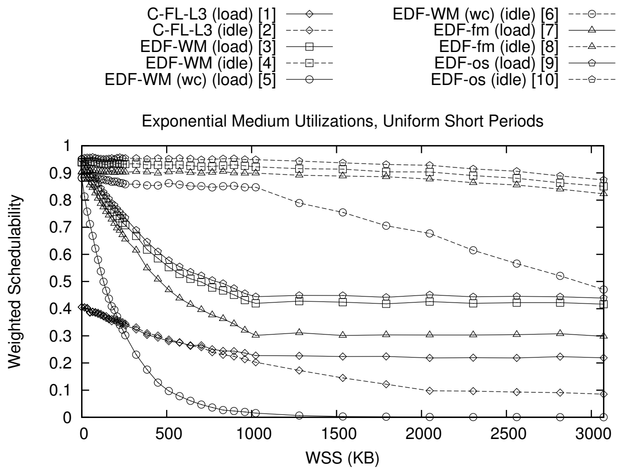

Figure 33: Exponential Medium Utilizations, Uniform Short Periods

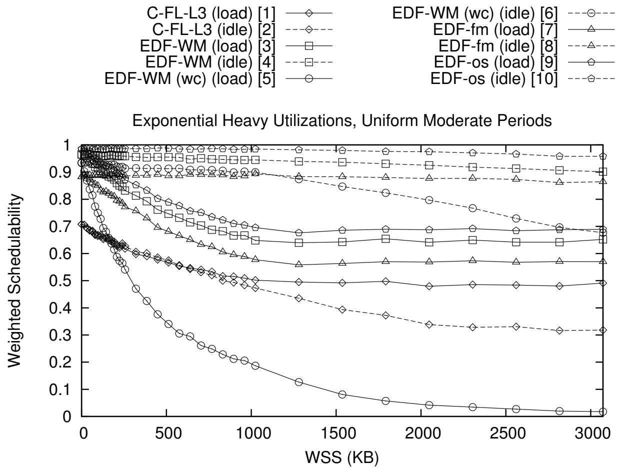

Figure 31: Exponential Heavy Utilizations, Uniform Moderate Periods

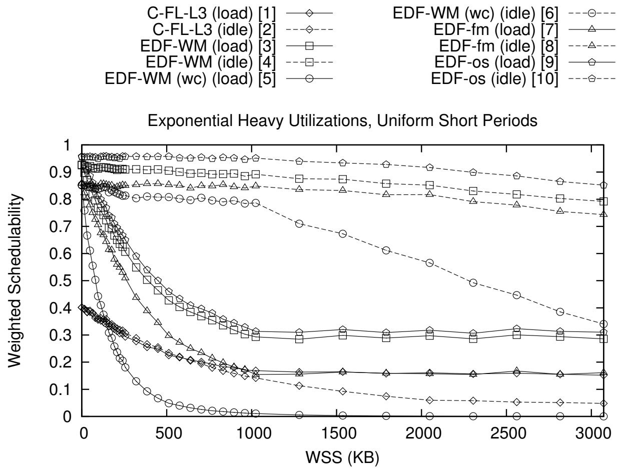

Figure 34: Exponential Heavy Utilizations, Uniform Short Periods

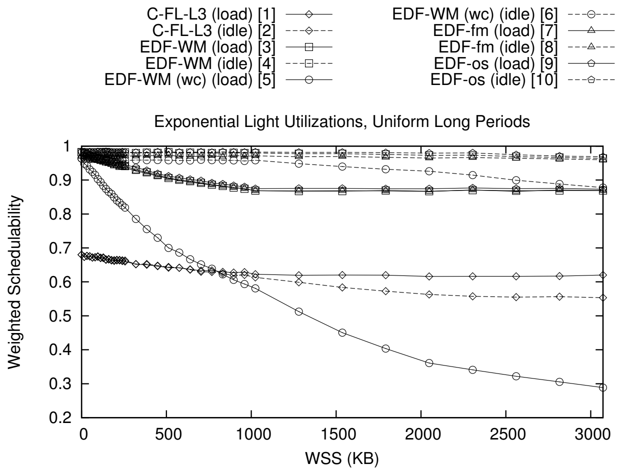

Figure 35: Exponential Light Utilizations, Uniform Long Periods

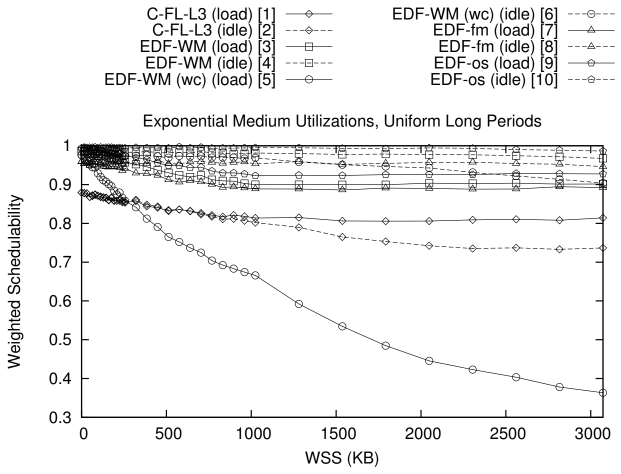

Figure 36: Exponential Medium Utilizations, Uniform Long Periods

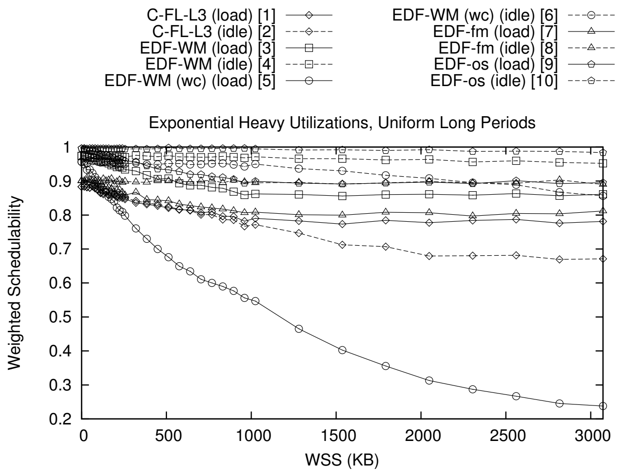

Figure 37: Exponential Heavy Utilizations, Uniform Long Periods

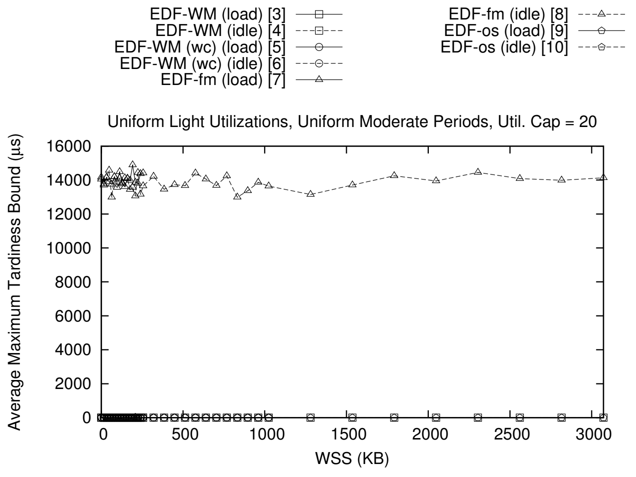

Figure 38: Uniform Light Utilizations, Uniform Moderate Periods, Util. Cap = 20

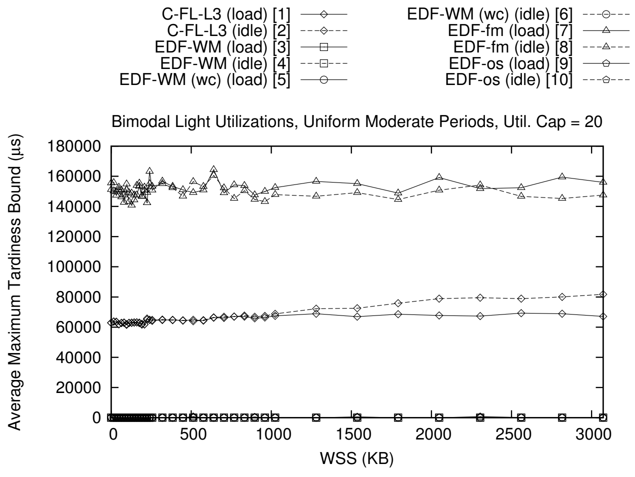

Figure 41: Bimodal Light Utilizations, Uniform Moderate Periods, Util. Cap = 20

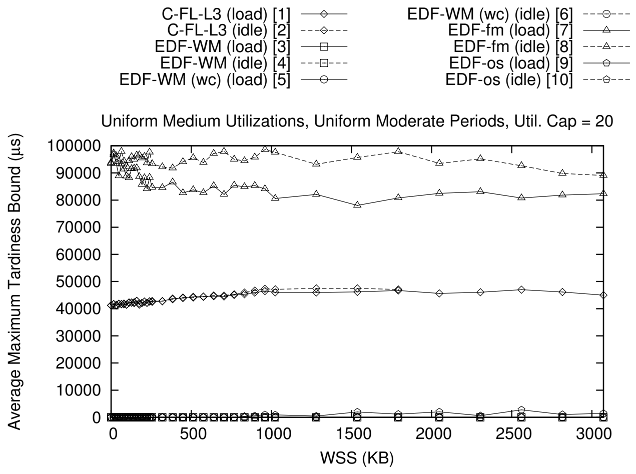

Figure 39: Uniform Medium Utilizations, Uniform Moderate Periods, Util. Cap = 20

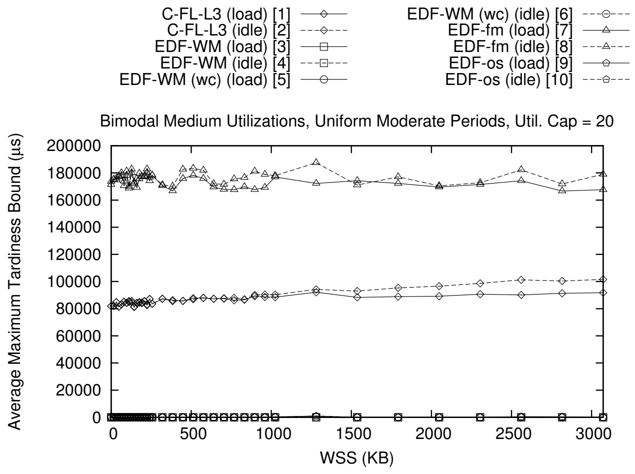

Figure 42: Bimodal Medium Utilizations, Uniform Moderate Periods, Util. Cap = 20

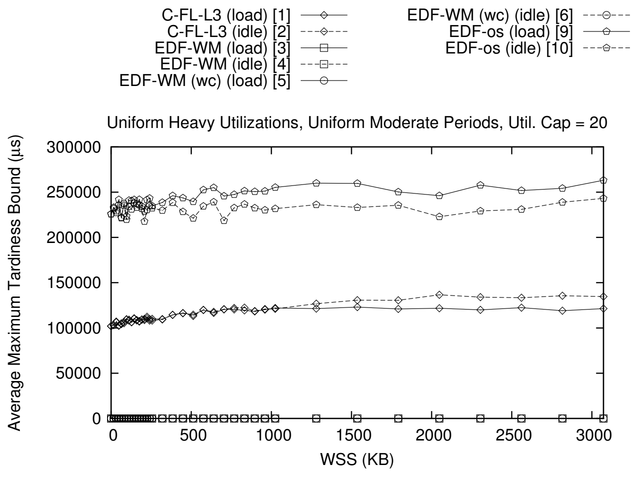

Figure 40: Uniform Heavy Utilizations, Uniform Moderate Periods, Util. Cap = 20

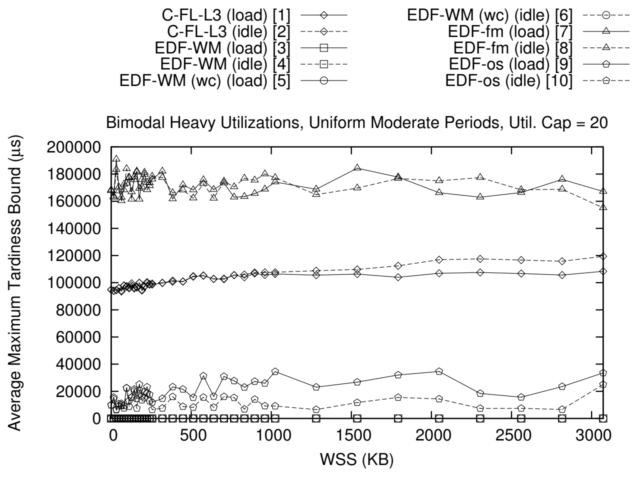

Figure 43: Bimodal Heavy Utilizations, Uniform Moderate Periods, Util. Cap = 20

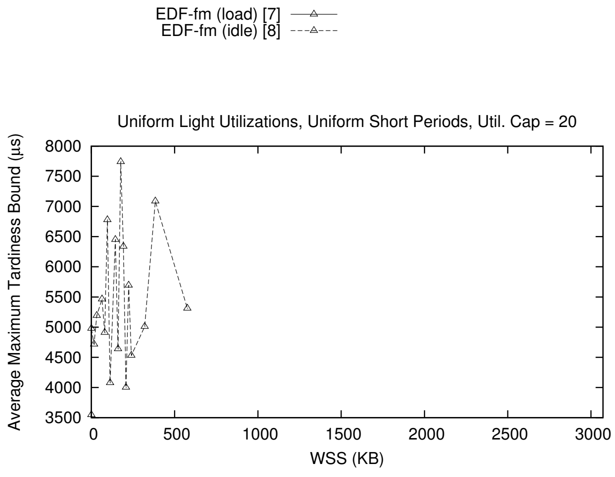

Figure 44: Uniform Light Utilizations, Uniform Short Periods, Util. Cap = 20

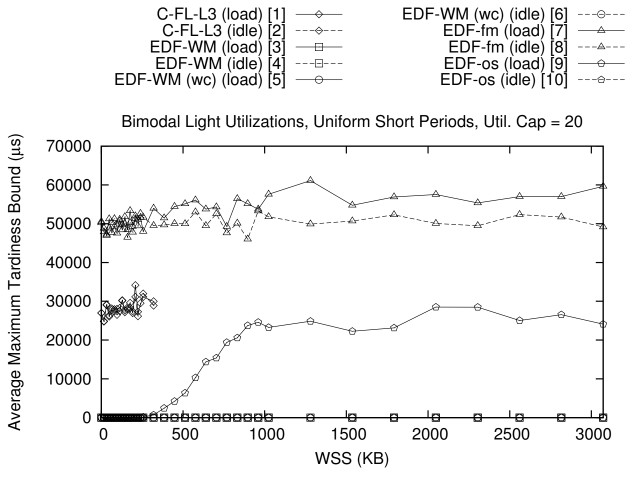

Figure 47: Bimodal Light Utilizations, Uniform Short Periods, Util. Cap = 20

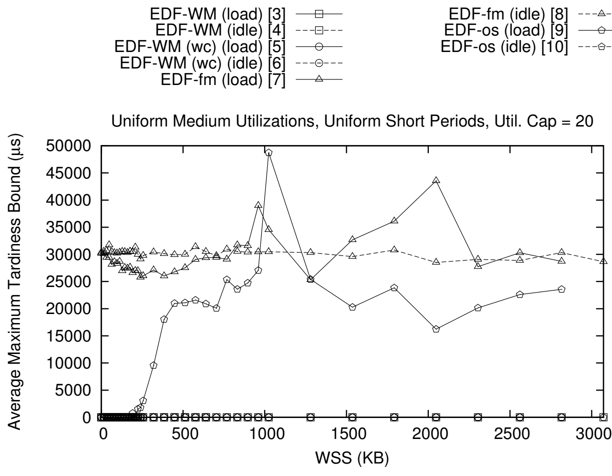

Figure 45: Uniform Medium Utilizations, Uniform Short Periods, Util. Cap = 20

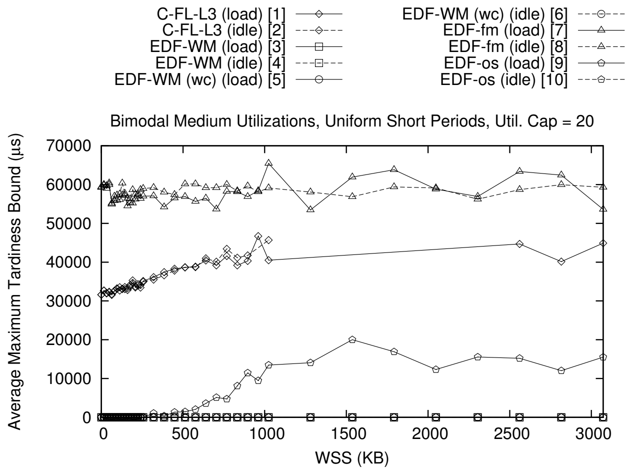

Figure 48: Bimodal Medium Utilizations, Uniform Short Periods, Util. Cap = 20

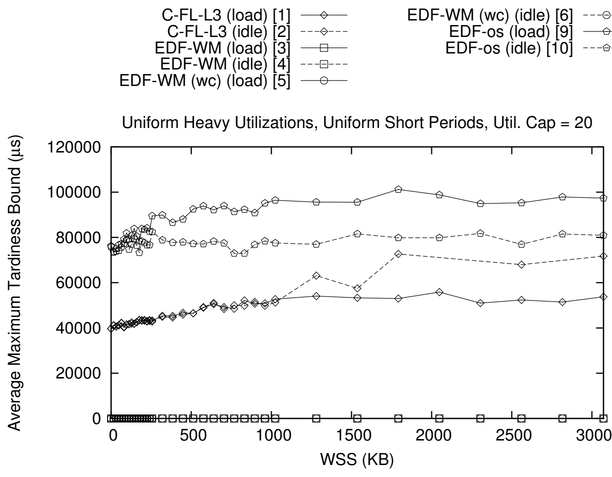

Figure 46: Uniform Heavy Utilizations, Uniform Short Periods, Util. Cap = 20

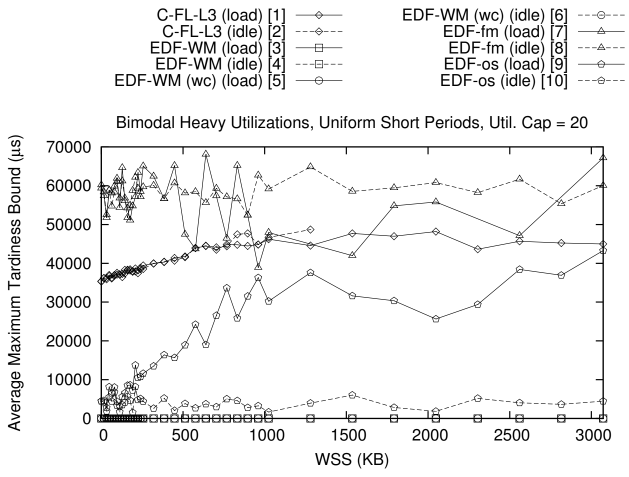

Figure 49: Bimodal Heavy Utilizations, Uniform Short Periods, Util. Cap = 20

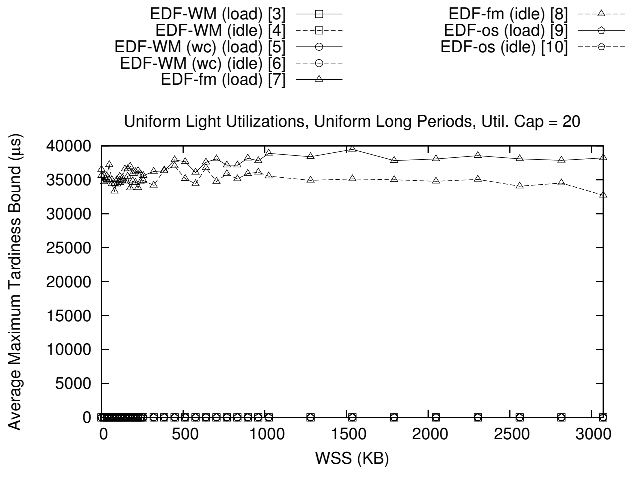

Figure 50: Uniform Light Utilizations, Uniform Long Periods, Util. Cap = 20

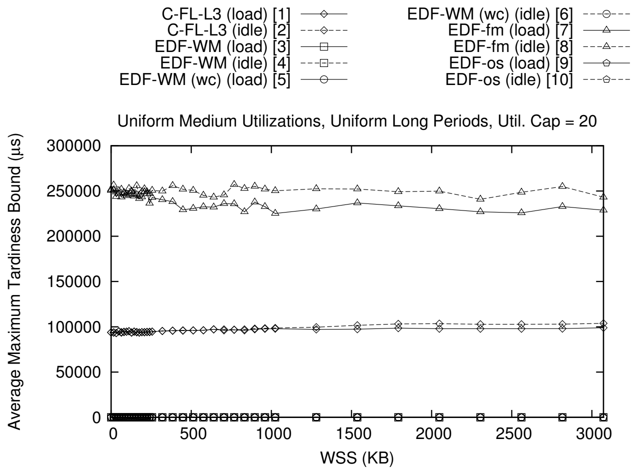

Figure 51: Uniform Medium Utilizations, Uniform Long Periods, Util. Cap = 20

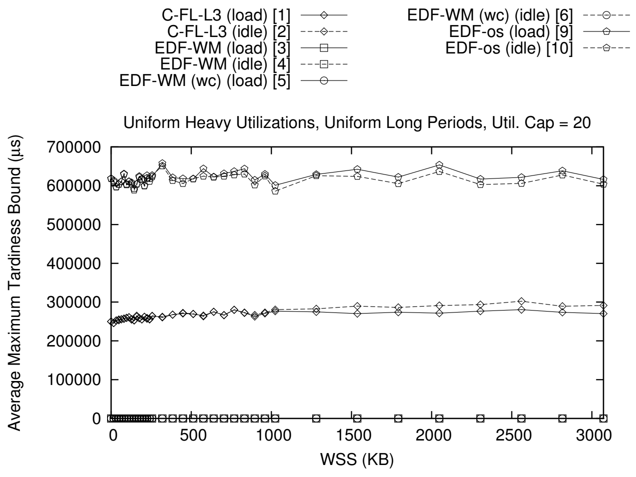

Figure 52: Uniform Heavy Utilizations, Uniform Long Periods, Util. Cap = 20

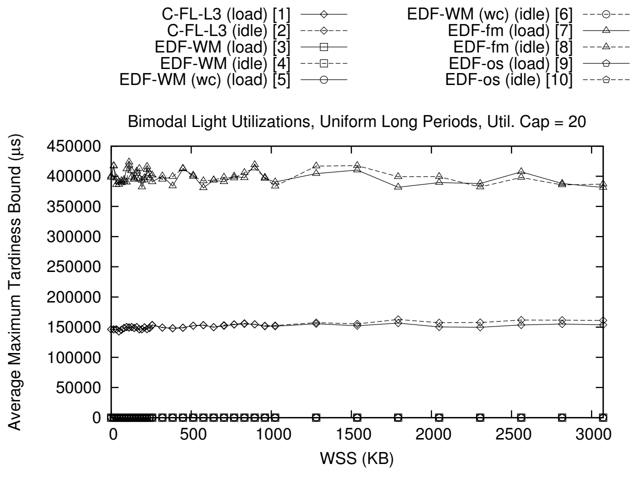

Figure 53: Bimodal Light Utilizations, Uniform Long Periods, Util. Cap = 20

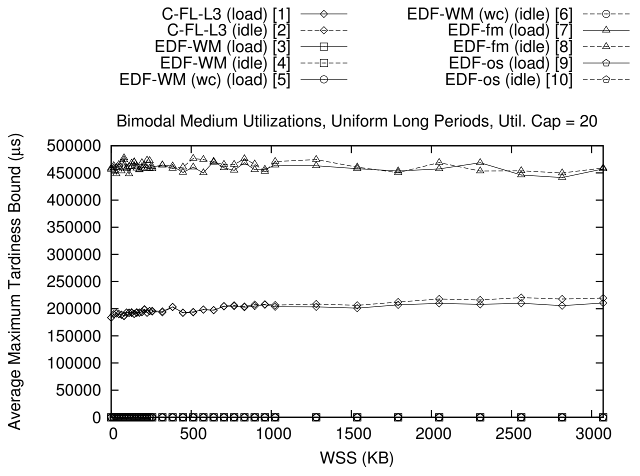

Figure 54: Bimodal Medium Utilizations, Uniform Long Periods, Util. Cap = 20

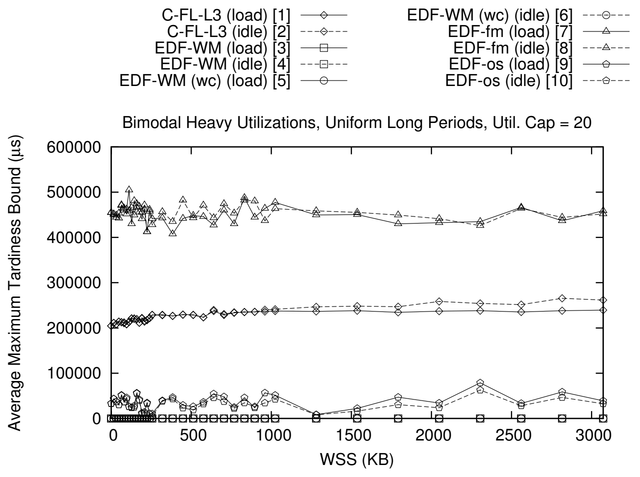

Figure 55: Bimodal Heavy Utilizations, Uniform Long Periods, Util. Cap = 20

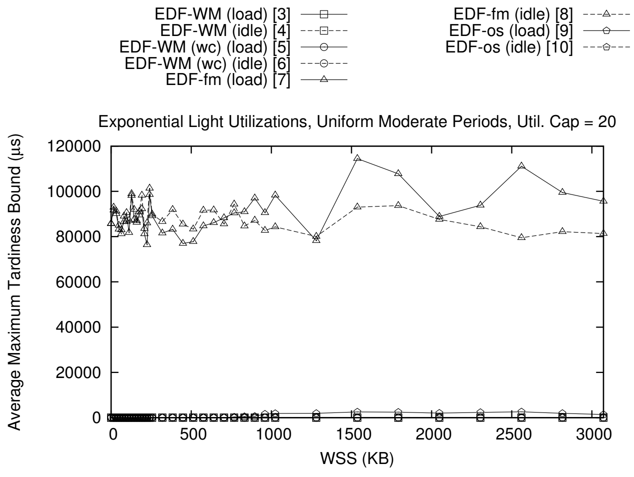

Figure 56: Exponential Light Utilizations, Uniform Moderate Periods, Util. Cap = 20

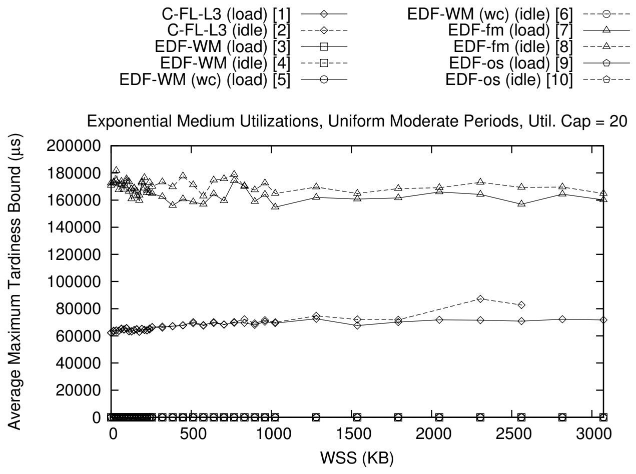

Figure 57: Exponential Medium Utilizations, Uniform Moderate Periods, Util. Cap = 20

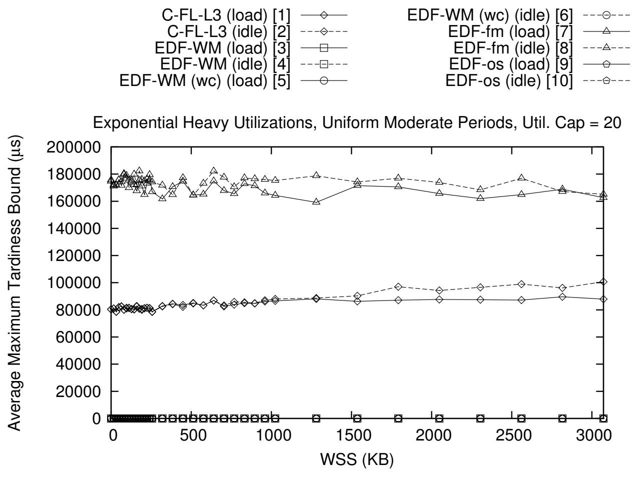

Figure 58: Exponential Heavy Utilizations, Uniform Moderate Periods, Util. Cap = 20

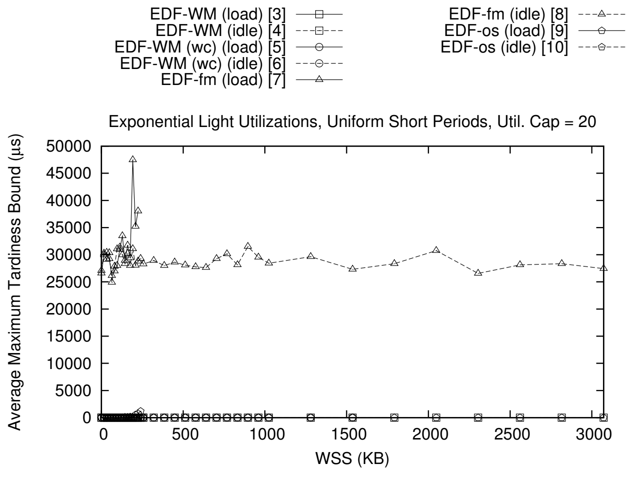

Figure 59: Exponential Light Utilizations, Uniform Short Periods, Util. Cap = 20

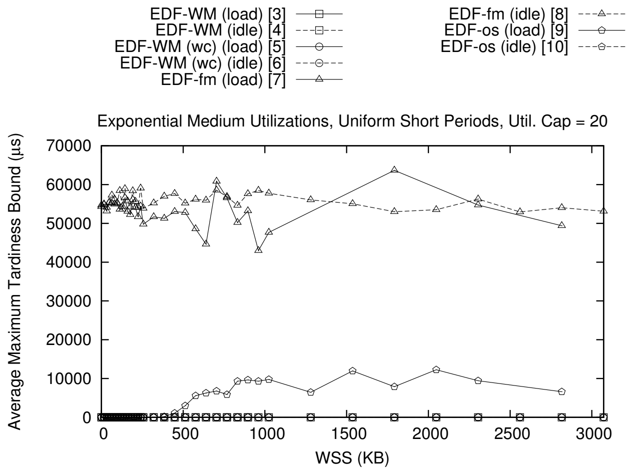

Figure 60: Exponential Medium Utilizations, Uniform Short Periods, Util. Cap = 20

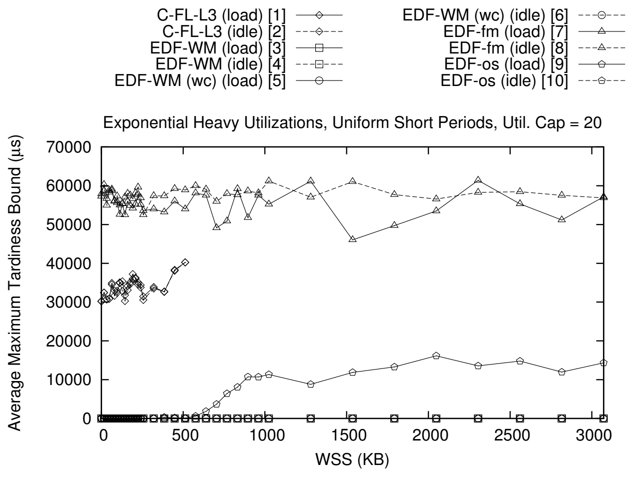

Figure 61: Exponential Heavy Utilizations, Uniform Short Periods, Util. Cap = 20

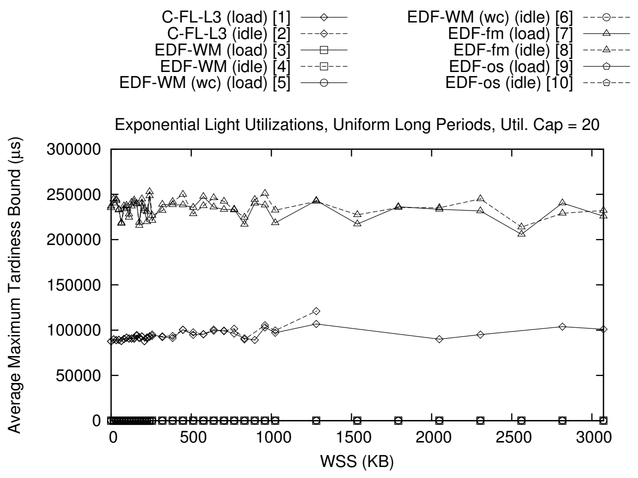

Figure 62: Exponential Light Utilizations, Uniform Long Periods, Util. Cap = 20

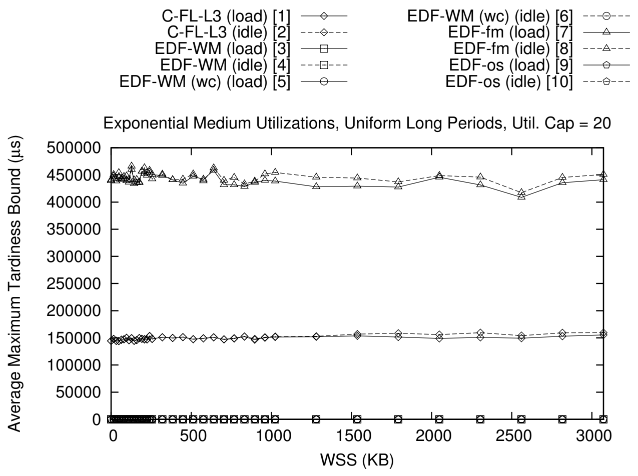

Figure 63: Exponential Medium Utilizations, Uniform Long Periods, Util. Cap = 20

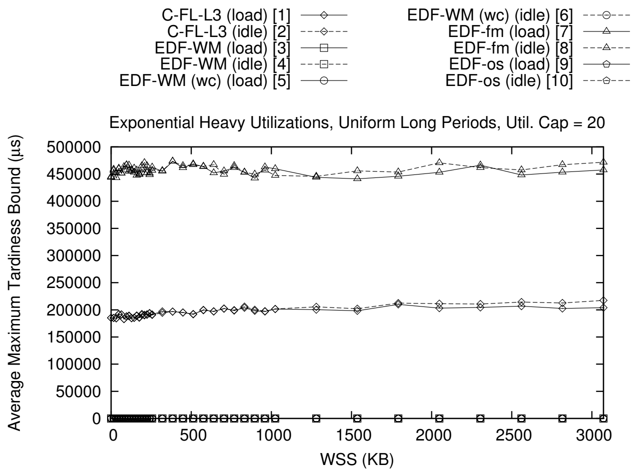

Figure 64: Exponential Heavy Utilizations, Uniform Long Periods, Util. Cap = 20

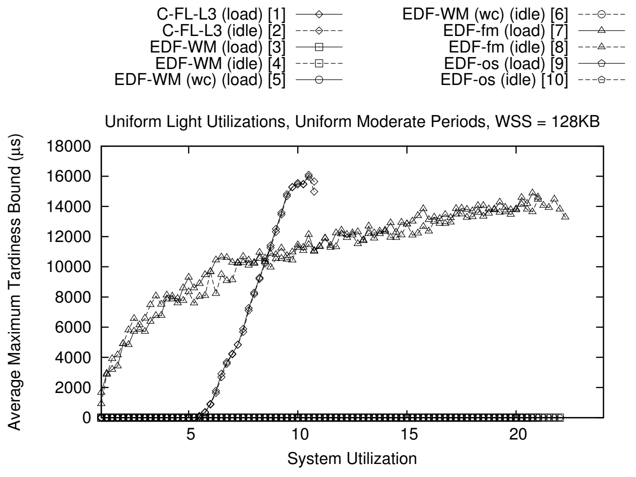

Figure 65: Uniform Light Utilizations, Uniform Moderate Periods, WSS = 128KB

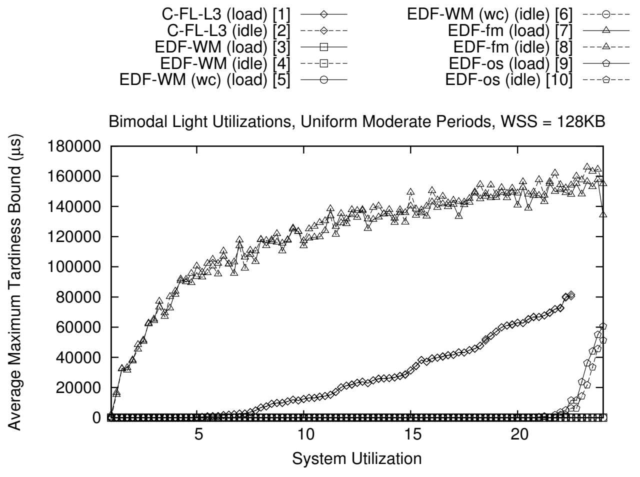

Figure 68: Bimodal Light Utilizations, Uniform Moderate Periods, WSS = 128KB

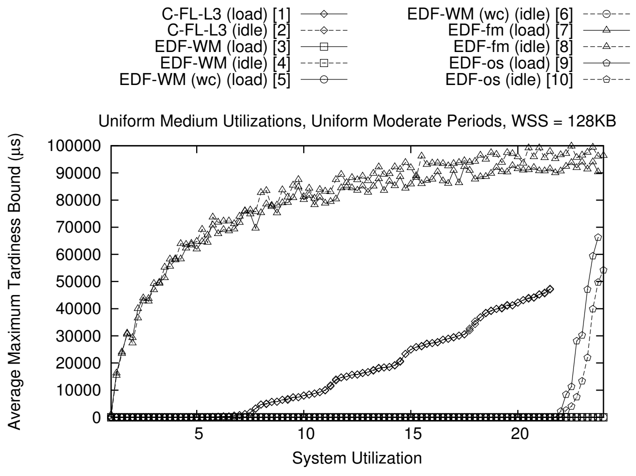

Figure 66: Uniform Medium Utilizations, Uniform Moderate Periods, WSS = 128KB

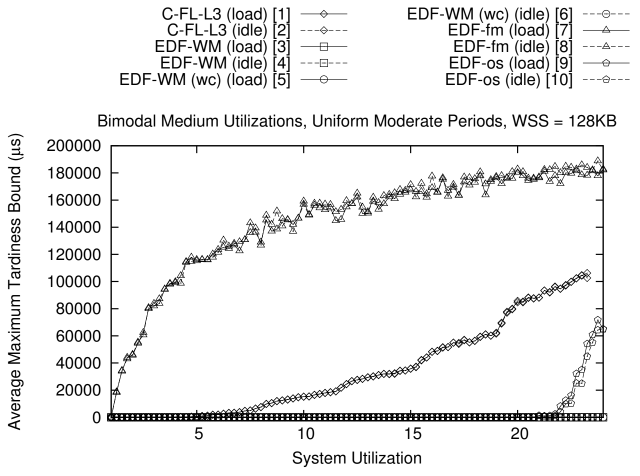

Figure 69: Bimodal Medium Utilizations, Uniform Moderate Periods, WSS = 128KB

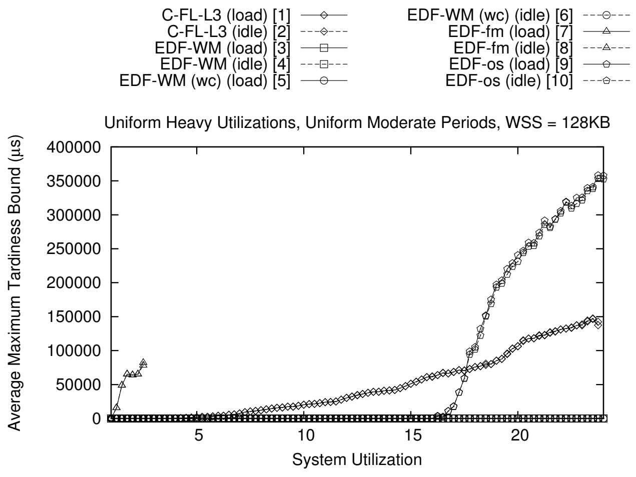

Figure 67: Uniform Heavy Utilizations, Uniform Moderate Periods, WSS = 128KB

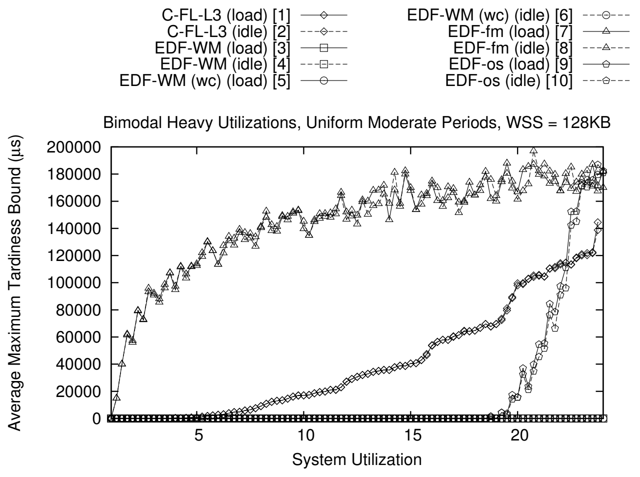

Figure 70: Bimodal Heavy Utilizations, Uniform Moderate Periods, WSS = 128KB

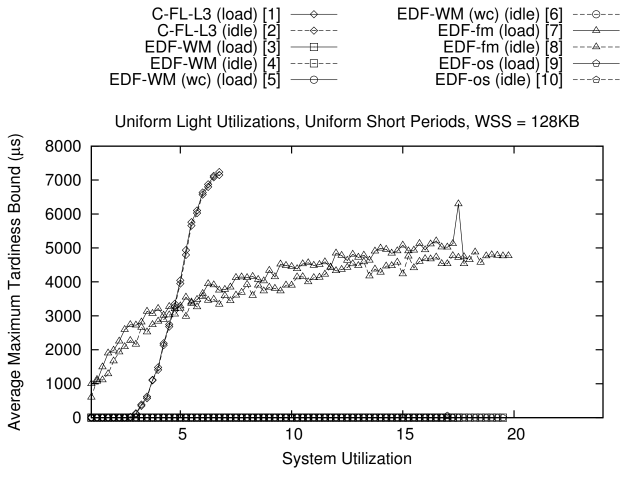

Figure 71: Uniform Light Utilizations, Uniform Short Periods, WSS = 128KB

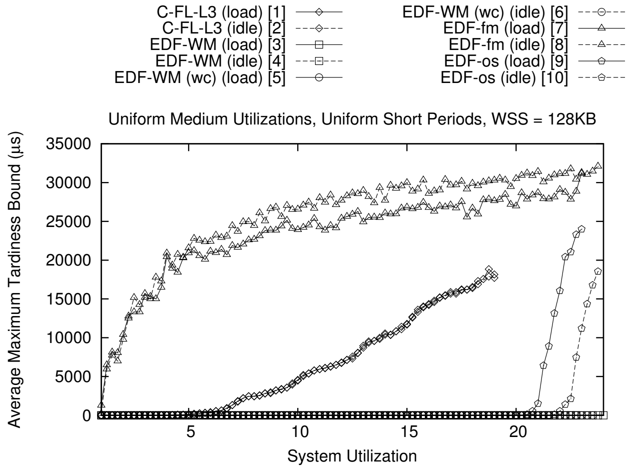

Figure 72: Uniform Medium Utilizations, Uniform Short Periods, WSS = 128KB

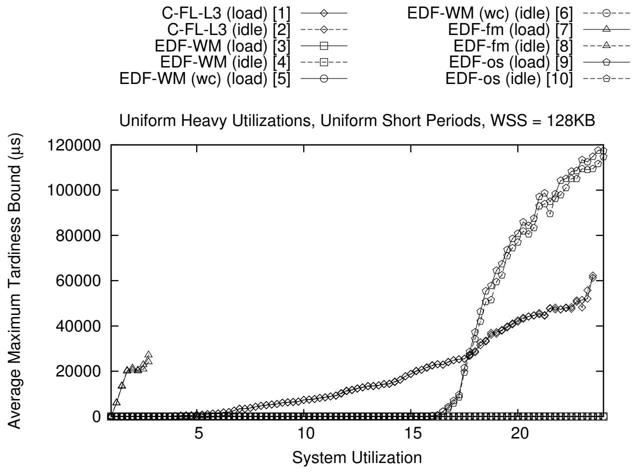

Figure 73: Uniform Heavy Utilizations, Uniform Short Periods, WSS = 128KB

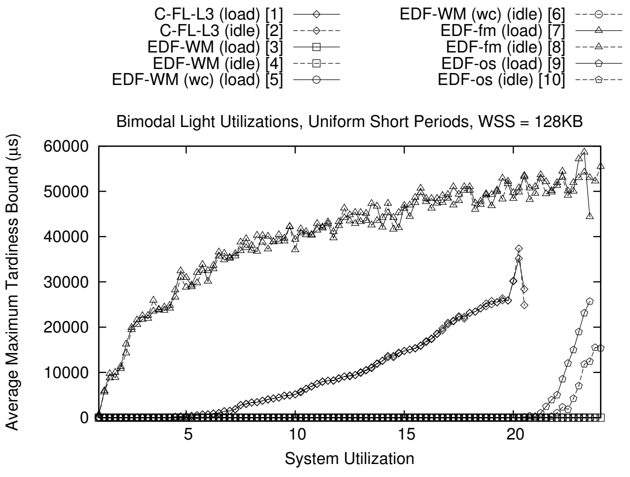

Figure 74: Bimodal Light Utilizations, Uniform Short Periods, WSS = 128KB

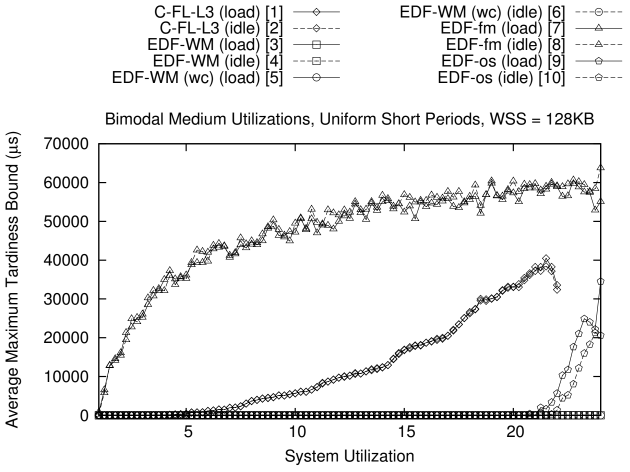

Figure 75: Bimodal Medium Utilizations, Uniform Short Periods, WSS = 128KB

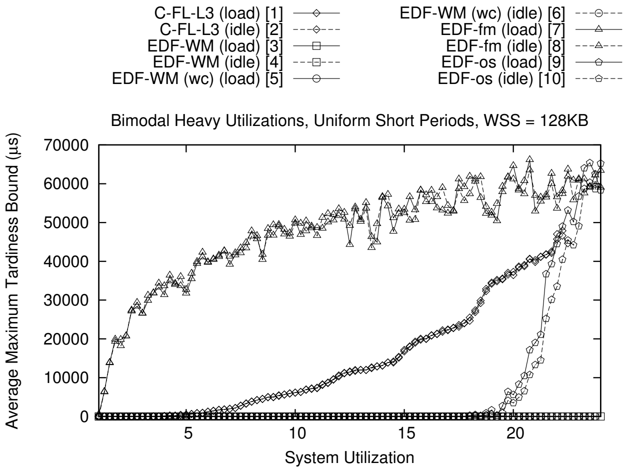

Figure 76: Bimodal Heavy Utilizations, Uniform Short Periods, WSS = 128KB

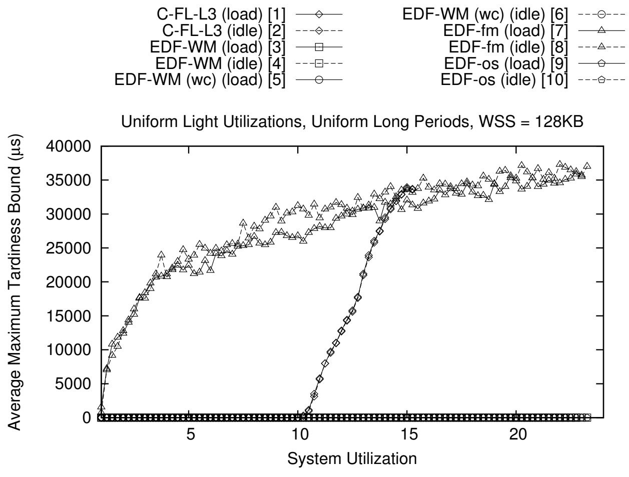

Figure 77: Uniform Light Utilizations, Uniform Long Periods, WSS = 128KB

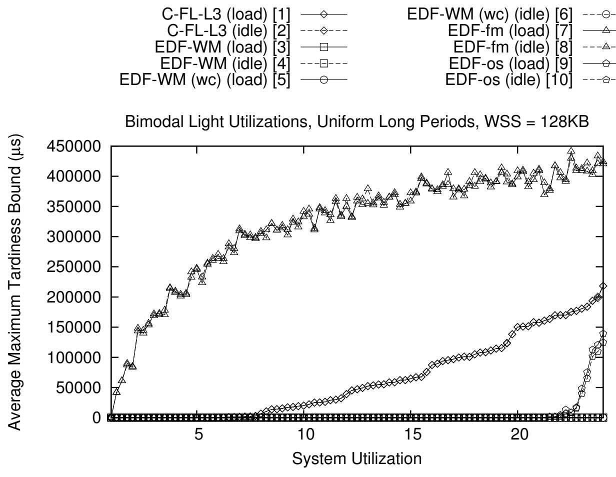

Figure 80: Bimodal Light Utilizations, Uniform Long Periods, WSS = 128KB

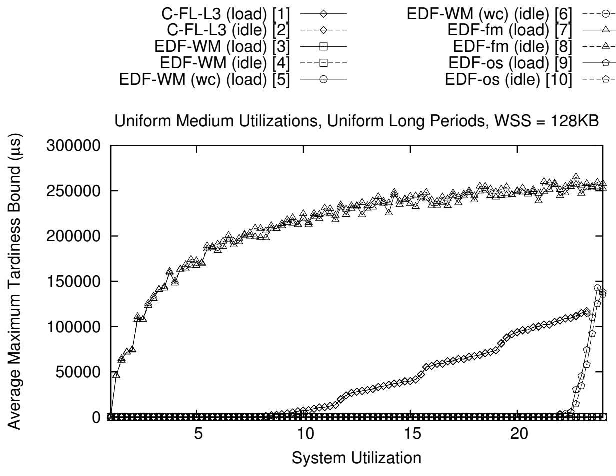

Figure 78: Uniform Medium Utilizations, Uniform Long Periods, WSS = 128KB

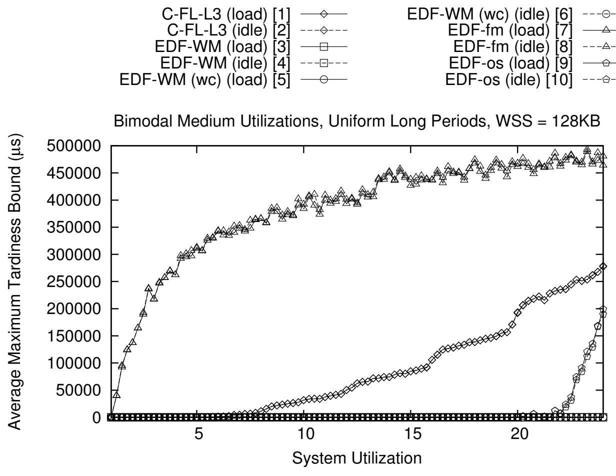

Figure 81: Bimodal Medium Utilizations, Uniform Long Periods, WSS = 128KB

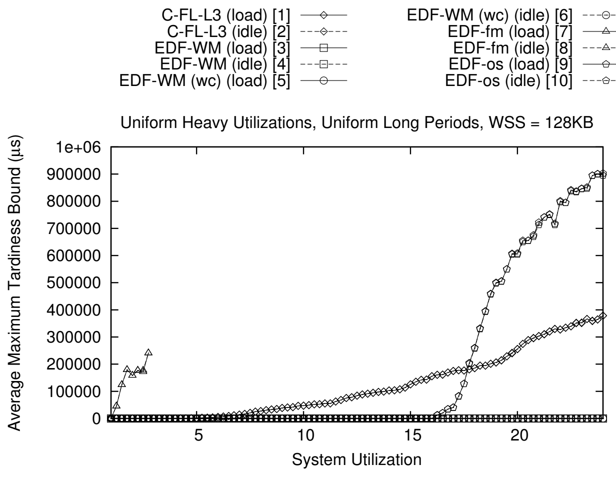

Figure 79: Uniform Heavy Utilizations, Uniform Long Periods, WSS = 128KB

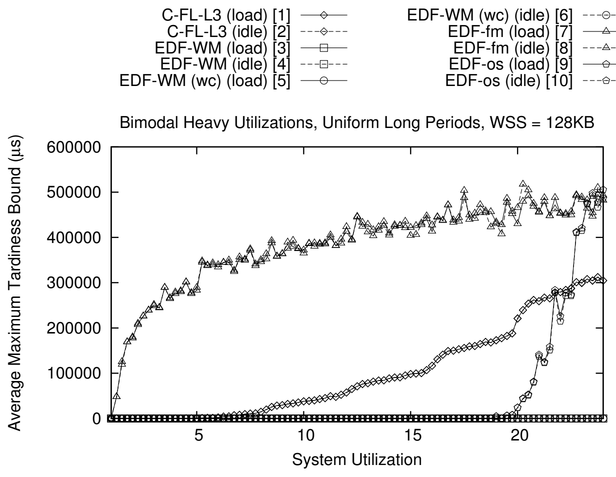

Figure 82: Bimodal Heavy Utilizations, Uniform Long Periods, WSS = 128KB

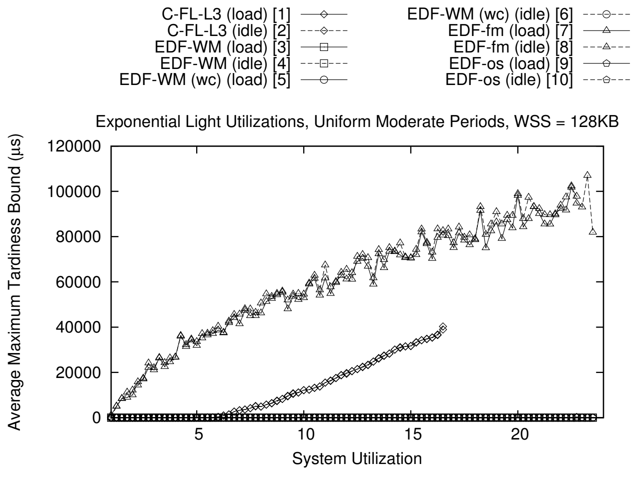

Figure 83: Exponential Light Utilizations, Uniform Moderate Periods, WSS = 128KB

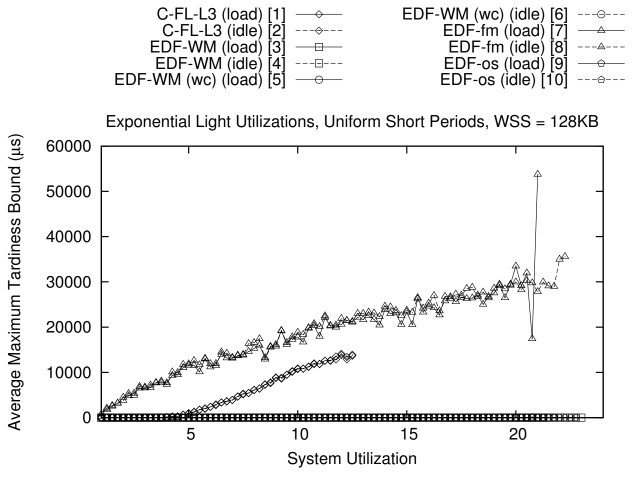

Figure 86: Exponential Light Utilizations, Uniform Short Periods, WSS = 128KB

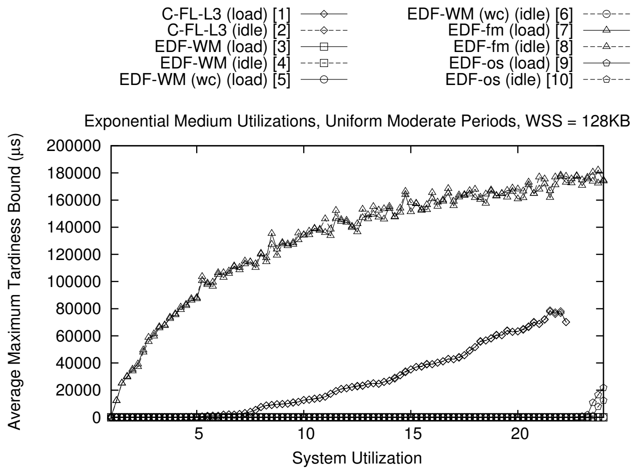

Figure 84: Exponential Medium Utilizations, Uniform Moderate Periods, WSS = 128KB

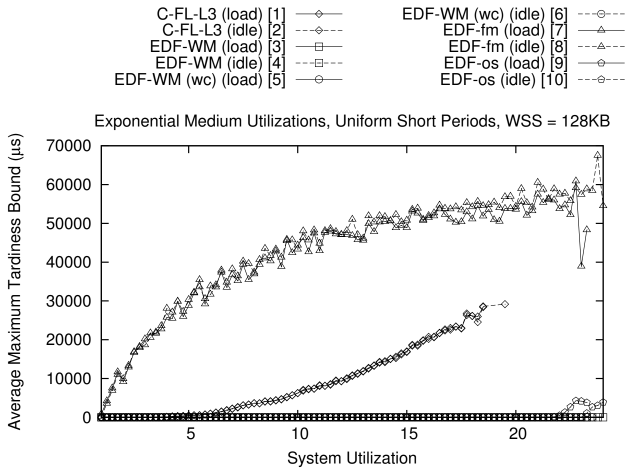

Figure 87: Exponential Medium Utilizations, Uniform Short Periods, WSS = 128KB

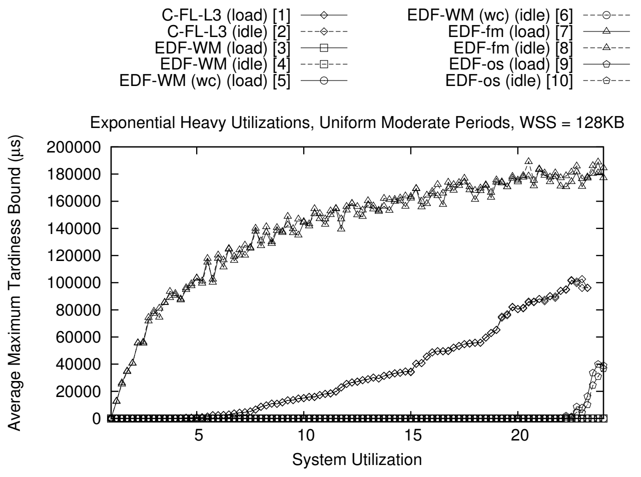

Figure 85: Exponential Heavy Utilizations, Uniform Moderate Periods, WSS = 128KB

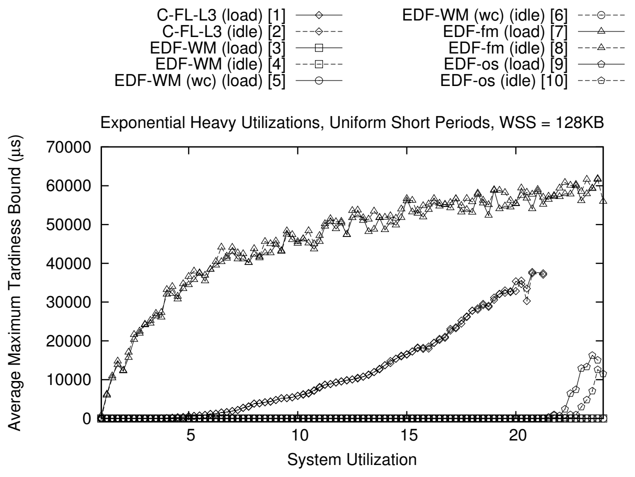

Figure 88: Exponential Heavy Utilizations, Uniform Short Periods, WSS = 128KB

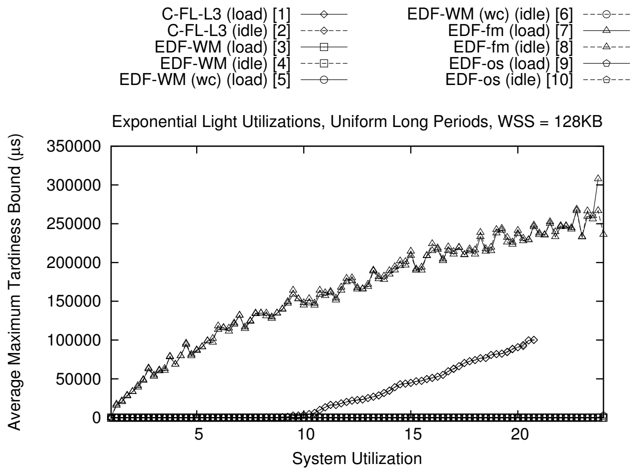

Figure 89: Exponential Light Utilizations, Uniform Long Periods, WSS = 128KB

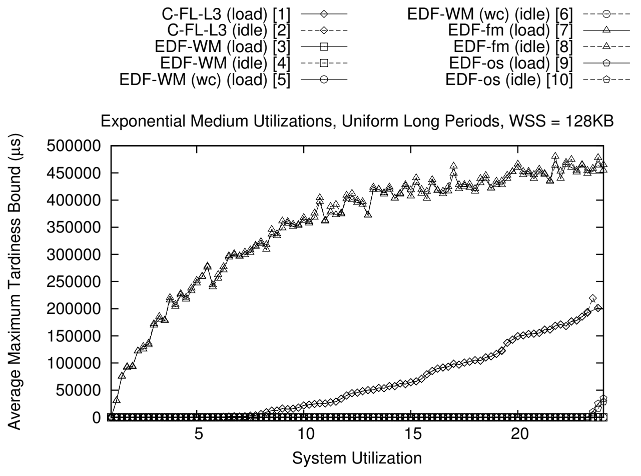

Figure 90: Exponential Medium Utilizations, Uniform Long Periods, WSS = 128KB

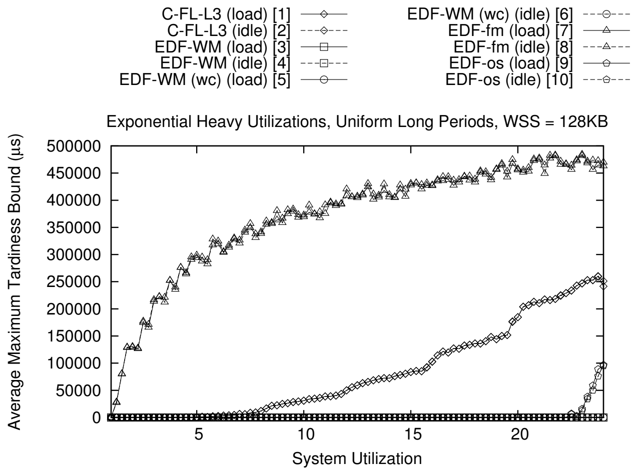

Figure 91: Exponential Heavy Utilizations, Uniform Long Periods, WSS = 128KB