Reviewer A:

The paper discusses the application of the period enforcer algorithm to segmented self-suspending tasks in a unicore/partitioned-multicore settings. In particular, the authors contrived multiple scenarios and task sets to exemplify situations where (1) schedulability is achieved without using the period enforcer but deadlines are missed if the enforcer is used, and (2) the enforcer is not compatible with multicore resource sharing mechanisms in the sense that it invalidates the analyses currently employed. As an extra contribution, the authors also show that converting sporadic segmented self-suspending tasks into single-segment deferrable tasks (i.e. tasks with release jitters) incurs extra pessimism in the analysis (which means that the original task set may be schedulable whereas its corresponding deferrable tasks set is not), and doing the conversion in polynomial time is an open problem.

The problem is relevant and interesting and I believe the paper fits very well in the scope of Lites. The structure of the paper is good and the presentation is excellent, both in terms of language, and general structure. The paper is easy and pleasant to read and presents very pedagogic content. I warmly recommend its publication.

Q:A.1: I only have a few very minor comments:
- p2, line 60: assumptions on control flow $\rightarrow$ assumptions on the task control flow
- p2, line 60: the segmented model is that is allows for $\rightarrow$ the segmented model is that **it** allows for
- p4, line112: I would say segmented self-suspending tasks rather than multi-segment tasks to remain consistent with the naming introduced in the system model.
- p7, line 212: to enrich the period enforcer with the following power, consider replacing power with "property" or "scheduling rule".
- p7, line 221: "it is immediately obvious that" $\rightarrow$ I would simply remove that part of the sentence since nothing is obvious with self-suspending tasks and particularly for the problem addressed in this paper.

Response: Thanks for the suggestions and comments. We have rephrased those parts accordingly.

Reviewer B:

This work makes the following observations (i) period enforcement is not strictly superior to scheduling without enforcement (ii) period enforcement is incompatible with existing analysis of suspension-based locking protocols.

The observations described above are valid. However, there are some unstated assumptions about the period enforcer in how the technique is being applied, which should be clarified in this paper.

Q:B.1: First, the period enforcer analysis only applies to the case where a task defers its entire execution time but not parts of it. Tasks that suspend after executing for a while should be considered as two or more tasks with their own worst-case execution time. It is known that if tasks are broken up into multiple segments with individual relative deadlines adding up to the original deadline, the schedulability is no longer guaranteed. For example, a task set (C,T,D), with tasks (4, 5, 5) and (2, 10, 10) is schedulable, whereas breaking up the (2, 10, 10) into (1.5, 10, 7.5) and (0.5, 10, 2.5) is not schedulable.

Section 3 should address the above clarification and the task τ_2 should be statically broken up before applying the period enforcer. It is true that transforming a task set by breaking up tasks into sub tasks with sub deadlines can lead to unschedulability on an otherwise schedulable task set.

Response: Reviewer B was partially correct. Since the period enforcer analysis only applies to the case where a task defers its entire execution time but not parts of it, we should create the corresponding deferrable task set. However, it is not necessary that we need to create those deferrable task sets before using the period enforcer algorithm. Since the computation segments of a segmented self-suspending task are independent from each other, we can simply use the period enforcer algorithm without knowing the corresponding deferrable task set. This is actually how we demonstrate the examples in Section 2 and Section 3. However, there is a risk that the resulting schedule may not be feasible. Therefore, one would need to verify the feasibility of the corresponding deferrable task set before applying the period enforcer algorithm. Such a step was assumed in [21], and we explain the difficulty in this paper. The period enforcement in the example in Section 3 is only used to enforce the periodic behavior of the 2nd computation segment of task τ_2. We do not specifically discuss the schedulability test over there. Our intention is to show that the period enforcer algorithm is not always superior to the original fixed-priority scheduling. That is, the enforcement makes this segment miss the deadline, but, in the fixed-priority scheduling there is no deadline miss. We leave the discussions about the task transformation and the schedulability test in Section 4. To be more precise about the above statement, we have rewritten Section 4 to answer two fundamental questions: First, how can we derive the corresponding deferrable task set efficiently? Second, how should we perform the schedulability test on the corresponding deferrable task set correctly?

Actions Taken: *Section 4 was completely rewritten to be more clear.* □

Q:B.2: Section 4 should be reworded. 'precisely converting a self-suspending task system into a corresponding single-segment deferrable task set is not an easy problem' $\rightarrow$ 'converting a self-suspending tasks system into a single-segment deferrable task set introduced pessimism due to period enforcement'. It is unclear how the Nelissen paper helps since it does not assume period enforcement. If each self-suspending task is broken into deferrable sub tasks (considering all the higher priority tasks as non-suspending), why is the Nelissen analysis needed? Also, It should be clarified as to why the problem of precisely computing W_k^j is useful in the first place since period enforcement would introduce pessimism anyway (as shown in Section 3).

Response: We do not fully capture the first sentence here from the reviewer. There is no additional pessimism due to the period enforcer algorithm. The pessimism is due to the assumption made by the period enforcer algorithm to allow only a set of deferrable tasks (that only suspend once at beginning). In the revision, we completely rewrote Section 4 to explain the gap between the segmented self-suspending task model and the corresponding deferrable sporadic task set. We need the analysis from Nelissen et al. to precisely break a self-suspending task into multiple deferrable subtasks so that we can calculate the worst-case release jitter of a computation segment. The new structure of Section 4 first explains why it is difficult to precisely calculate the release jitter of the computation segments and then explains that it is rather important to also perform the

schedulability test correctly. We show that the schedulability test can be wrong if the test is not done well.

Q:B.3: Section 5 should also include the scenario where period enforcement is only applicable to the normal execution segments. It seems practical to not enforce critical sections since they are usually short in most systems i.e. the back-to-back execution overhead from most critical sections in real-world scenarios should be minimal. The case 3 that should be discussed is period enforcement applied to execution segment after the critical section, without enforcing the critical section and just including its back-to-back execution penalty in the worst-case analysis.

Response: leave to Björn

Reviewer C:

Q:C.1: This reviewer was not able to download a complete version of [20] including the figures (the .ps available online is highly corrupted). Please provide a precise link to the full version of this paper or a way to access to it.

Response: We are not able to provide a complete version of the paper since the paper is not from us. We also used the same version (without figures) to study the period enforcer algorithm. In the revision, we have decided to detail more steps to improve the consistency of our paper.

Reviewer D:

The period enforcer algorithm (proposed by Rajkumar [20]) was designed to reduce the scheduling penalty caused by self-suspending tasks. This paper presents three important observations regarding the applicability of period enforcer algorithm and its schedulability analysis to multi-segment self-suspending tasks. First, it is shown that period enforcement may cause deadline miss in multi-segment self-suspending tasks that are schedulable without period enforcement. The original analysis of period enforcer algorithm [20] shows that if a set of single-segment (deferrable) tasks is schedulable, then the task set is also schedulable under period enforcement. Therefore, to apply this result from [20] to a set of multi-segment self-suspending tasks, a transformation from multi-segment to single-segment task set is needed. Second, it is shown that such transformation is quite difficult (i.e., currently only exponential time solution is known). Third, it is shown that period enforcer algorithm requires new analysis of existing suspension-based locking protocol to compute blocking factor. In other words, period enforcer algorithm is incompatible with all existing analyses of suspension-based locking protocols. All these three (negative) observations show the limitations of the applicability of period enforcer algorithm to multi-segment self-suspending task set. These limitations are demonstrated using examples that are easy to follow (Section 3, 4 and 5).

Q:D.1: This paper is heavily dependent on the assumptions and analysis of the original period enforcer algorithm in [20]. Although the authors highlighted some of the assumptions and important result (e.g., Theorem 5) from [20], it may be difficult for readers to understand this paper without reading in details the analysis from [20]. The reviewer had difficulty in understanding the discussion in section 2.3 and 2.4. After the reviewer read [20], the discussion in section 2.3 and section 2.4 becomes clearer. Important details from [20] need to be added to this paper to make it

more readable and self-contained.

Response: leave this to Björn

Q:D.2: There are some terms in section 2.3 and 2.4 that the authors have used assuming that readers are already familiar with those terms. For example, the terms deferrable task, task set transformation, single-segment deferrable tasks, non-self-suspending sporadic task, etc. may be formally defined before using them. For example, the set of transformed non-self-suspending tasks subject to release jitter (section 2.4) essentially means a set of deferrable tasks. Since Theorem 5 uses the term deferrable task, the relationship between transformed task set and deferrable task set may be explicitly stated for better understanding.

Response: leave this to Björn as well

Q:D.3: Theorem 5 (page 5) mentions about worst-case condition. What does it mean by worst-case condition for deferrable tasks? Paper [20] explains the worst-case. The authors could explain the worst-case or restate the theorem in a different way so that reader does not need to seek for the worst-case in [20].

Response: leave this to Björn as well. message from JJ: Can you think how to improve that? I think a table would be nice. We may need to put more words from Rajs paper.. But, lets assume that Raj decides to take the paper down from his website. Then, no one knows what we are talking about precisely. So, maybe it is a good idea to make Section 2 more self-contained?

Q:D.4: Is there any way to quantify (theoretically or using simulation) the schedulability loss/gain using period enforcer algorithm for multi-segment self-suspending tasks? The reviewer is curious to learn the impact of the negative (theoretical) results of period enforcer algorithm on randomly generated multi-segment self-suspending task sets.

Response: We are not able to provide the (theoretical or simulated) loss/gain due to the following reasons. 1) We are not able to convert a segmented self-suspending task set into the corresponding deferrable task set efficiently and without any information loss. In Section 4.1, we explain the task set transformation for a special case. If there are multiple self-suspending tasks, there is no existing solution to exactly construct the deferrable task set. 2) There is no clear evidence on how such schedulability tests for the corresponding deferrable task set should be designed precisely. We have demonstrated in Section 4.2 that all the deferrable tasks (even constructed from the same self-suspending task) should be considered for analyzing the interference. But, this implies that the worst-case behaviour of the last computation segment can be very pessimistic. Based on the example provided in Section 4.2, one can imagine that the calculation of the deferrable time already considers the earlier computation segments and the calculation of the worst-case interference has to account for the earlier computation segments as well (even if there is no overlap). We are not sure whether this is due to the design or due to the pessimistic analysis. Please refer to Section 4.2 for this. 3) There are also other methods that have been developed recently to handle the self-suspensions, and we are not sure what to compare with.

Actions Taken: *Section 4 has been completely rewritten to explain the difficulty.* □

Q:D.5: Based on the notes of this paper, it is not clear whether one should consider period enforcer algorithm or not for self-suspending tasks.

Response: From our note, we can only support the period enforcer algorithm as stated in the conclusion: "*Nevertheless, Theorem 5 in [21] could be useful for handling self-suspending tasks (that do not use suspension-based locks) if there exist* efficient *schedulability tests for the corresponding deferrable task systems or the period enforcer algorithm. However, such tests have not been found yet and the development of a precise and efficient schedulability test for self-suspending tasks remains an open problem.*" However, there are several drawbacks that were never discussed earlier, as pointed out at the end of Section 1. The period enforcer algorithm can still be potentially used with cares.

Q:D.6: Why the dynamic self-suspension model is presented and compared with segmented model in section 2.1? It is not clear why period enforcer algorithm applies only to the segmented model. Is there any reference or argument to support this claim in line 61-62?

Response: There was no reference to support this claim. We explain the reason in Section 2.3 in the revision.

Actions Taken: *The following paragraph is added to Section 2.3. Note that it is not possible to convert a dynamic self-suspending task into the notion of deferred execution defined in [21], since the definition of a dynamic self-suspending task creates dynamic execution amounts for different jobs of a task. A simple example can explain why the period enforcer algorithm is not compatible with the dynamic self-suspending task model. Suppose that the system has only one task with execution time* $C_1 = 1$, $S_1 = 1$, *and* $D_1 = T_1 = 2$. *The first job of task* τ_1 *arrives at time* 0, *suspends itself for one time unit, and then executes for one time unit. The second job of task* τ_1 *arrives at time* 2, *first executes for* 0.5 *time unit, then suspends for* 1 *time unit, and executes for* 0.5 *time unit. With the period enforcer algorithm, the second job of task* τ_1 *starts its execution at time* 3 *and will clearly miss the deadline at time* 4. □