TOWARDS EFFICIENT ON-LINE SCHEDULABILITY TESTS FOR ADAPTIVE NETWORKED EMBEDDED REAL-TIME SYSTEMS

Klaus Becker, Marc Zeller and Gereon Weiss
Fraunhofer Institute for Communication Systems ESK, Hansastrasse 32, 80686 Munich, Germany
{*klaus.becker, marc.zeller, gereon.weiss*}*@esk.fraunhofer.de*

Keywords: Schedulability Tests, Networked Embedded Systems, Adaptive Systems, Real-time.

Abstract: In networked embedded systems, runtime adaptive software promises an increase of flexibility, fault tolerance and extensibility. Often, this requires that software components have to be allocated dynamically to execution platforms at runtime. Hence, the platforms have to execute dynamically changing task sets. However, in real-time systems, a task set cannot be executed without previously checking its schedulability w.r.t. given timing constraints. Therefore, it has to be determined, whether or not the existing task set would be still schedulable including newly arriving tasks. In this paper, we propose a schedulability test algorithm for such systems, exploiting the situation of adding a new task to an existing task set. Therefore, we adapt existing schedulability tests to exploit the specific acceptance test use case. The benefits of the developed adaptations are shown by experimental investigations.

1 INTRODUCTION

In recent years, software has been increasingly integrated into a wide range of industrial application domains, while such embedded systems have experienced an enormous growth of complexity. The networked embedded software interacts with mechanical parts of the system and offers the bulk of the functionality. Inside these systems, often the contained embedded software components are distributed over multiple execution platforms and interact with each other over communication infrastructure to solve different types of tasks collaboratively. In today's networked embedded systems, the allocation of single software components onto the different execution platforms is mostly designed statically and does not change during runtime. In opposition to these static systems, a runtime adaptive system would provide several advantages (Oreizy et al., 1998). Examples are the possibility to react dynamically on hardware or software errors (*Self-Healing*) (Streichert et al., 2008), and the ability to start and stop software components during runtime, so that only the software components are active at a given time that are currently required. This allows an optimal usage of hardware resources, in order to minimize the amount of required hardware, which also results in minimizing the energy consumption (*Self-Optimization*) (Weiss et al., 2011).

In order to take advantage of these facts in a networked embedded system, it is necessary to be able to add new tasks to execution platforms during runtime or to migrate running tasks from one platform to another (*Structural Adaption* (McKinley et al., 2004)) (Zeller et al., 2011). However, before adding a new task to an execution platform, it is essential to keep track of given timing requirements, which have to be fulfilled by each of the tasks. A schedule has to be determined for the execution of the currently active tasks, so that no task violates its timing constraint, also called deadline. If these requirements cannot be ensured, a new task is not allowed to run on the platform. Thereby, the existing tasks can still meet their deadlines, assuming that this was the case before. Systems possessing those timing constraints are generally called real-time systems. The scheduling criteria are the priorities that are assigned to the tasks. For many industrial application domains of networked embedded systems, like automotive, these priorities are fixed and do not change during runtime. However, usually the priorities are defined by the system designer and not via a specific method like the rate monotonic (RM) priority assignment, introduced in (Liu and Layland, 1973).

In this paper, we analyze schedulability tests for sets of independent preemptive tasks with fixed priorities, blocking times due to resource constraints, and release jitters. The analyses are performed with the intention to provide a schedulability test, which is applicable on-line in adaptive embedded real-time systems. Therefore, existing approaches for schedulability tests are adapted in order to be usable as on-line acceptance tests for newly arriving tasks at runtime. Moreover, they are adapted to be compatible to the mentioned task model and to support tasks that have arbitrary priorities. Afterwards, the investigated algorithms and their adaptations are evaluated by means of their efficiency for randomly generated task sets. Also the effect of the used priority assignment is evaluated, providing results about the differences in assigning fixed priorities arbitrarily or rate monotonically.

However, only local acceptance tests are analyzed in the scope of this paper. No holistic schedulability tests that provide an end-to-end analysis including message transfer times are considered. Though, messages could be integrated easily into the shown approaches. It is shown in (Pop et al., 2003) and (Lei et al., 2004) that during schedulability tests, messages can be treated like non-preemptive tasks and communication channels can be treated a hardware platforms.

The following section gives a brief overview over state-of-the-art schedulability test algorithms.

2 RELATED WORK

Schedulability tests are specifically designed for the type of task sets, for which they are applied. In this paper, we assume task sets with fixed priorities. One way to assign fixed priorities is to assign them related to the task periods. If the periods are equal to the deadlines, this is called the *rate monotonic* (RM) priority assignment. This is shown to be optimal in (Liu and Layland, 1973), meaning that no other fixed priority assignment can result in a higher degree of schedulability. If the deadlines are smaller than the periods, the *deadline monotonic* (DM) priority assignment is optimal (Audsley et al., 1991).

In (Liu and Layland, 1973) also an approach to test the schedulability of task sets having RM priorities is introduced. This test is based on the total processor utilization $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$, where C_i is the *Worst Case Execution Time* (WCET) and T_i is the period of a task τ_i. It is shown that task sets with RM priorities are schedulable, if $U \leq n \cdot \left(2^{\frac{1}{n}} - 1\right)$. This test is only sufficient but not necessary, because there may exist task sets Γ_n with higher utilizations than $n \cdot \left(2^{\frac{1}{n}} - 1\right)$, which are schedulable, but are wrongly categorized as non-schedulable by the test. However, systems with arbitrary fixed priorities cannot be tested via this mechanism. A sufficient test for those systems was introduced in (Bini and Baruah, 2007). This test is not based on the processor utilization, but on the determination of an upper bound of the *Worst Case Response Time* (WCRT) R_i of any task $\tau_i \in \Gamma_n$ and its comparison to the respective relative deadline D_i.

A sufficient and necessary (and thereby exact) test for task sets with fixed priorities was introduced in (Joseph and Pandya, 1986), called the *Response Time Analysis* (RTA). This schedulability test observes every single task and calculates the interferences to the current task by all higher or equal prioritized tasks inside a specific time interval. Notice that the algorithm assumes that all tasks are released synchronously, because this release corresponds to the so-called *critical instant*, which leads to the WCRTs of all tasks, as shown in (Liu and Layland, 1973). The calculation is an iterative progress, because if interferences are found inside the current interval, the interval increases and new interferences may appear later. Via this, the latest possible finishing time of every task can be calculated and it can be proven, whether this lies prior to the respective deadline. If this is the case, the analyzed task is schedulable and the next task can be analyzed. Under the assumption of existing maximum blocking times B_i (while using an appropriate resource access protocol) and maximum release jitters J_i, the RTA can be described in a formal way by the following recurrence relation, introduced in (Audsley et al., 1993):

$$\begin{aligned} R_i^{(0)} &= B_i + C_i \\ I_i^{(k)} &= \sum_{\forall j \in hep(\tau_i)} \left\lceil \frac{R_i^{(k)} + J_j}{T_j} \right\rceil C_j \\ R_i^{(k+1)} &= B_i + C_i + I_i^{(k)} \end{aligned} \tag{1}$$

The calculation has to be continued, as long as the response time $R_i^{(k+1)} \neq R_i^{(k)}$ and $R_i^{(k+1)} \leq D_i - J_i$. As soon as R_i converges to a constant value, or R_i exceeds $D_i - J_i$, the calculation can be stopped. The subset $hep(\tau_i)$ corresponds to the subset of tasks that have a priority higher than or equal to the analyzed task τ_i. This algorithm is exact for independent preemptive tasks with fixed priories and without offsets. However, the runtime is deterministic but has a quite poor performance, compared to non-exact sufficient tests. Because of this, there exist several investigations about how to increase the performance of the RTA. An overview about state of the art techniques is given in (Davis et al., 2008). These techniques are based on increasing the initial value $R_i^{(0)}$ of the RTA,

thereby less iterations are required, which decreases the required runtime. Some of these techniques still allow an exact calculation of the WCRTs and other do not support an exact calculation, but still provide an exact Boolean answer about the schedulability. Because the latter initial values result in a higher efficiency and the Boolean answer satisfies all requirements of an on-line acceptance test, these values are used in the further proceeding of this paper.

It is shown by empirical investigation in (Davis et al., 2008) that using the maximum value of the following theorems 1, 2 and Equation (2) as initial value for the RTA leads to the best overall performance, if a Boolean answer is enough. Both theorems provide an upper bound on the WCRT R, while the equation provides a lower bound.

Theorem 1. *Using the initial value* $(D_i - J_i) - R_{i-1}^{UB}$, *the recurrence relation converges to* R_i^{UB}, *where* $R_i \leq R_i^{UB} \leq D_i - J_i$, *if* τ_i *is schedulable. Though, here it is assumed that* τ_{i-1} *is schedulable and therefore* $R_{i-1}^{UB} \leq (D_{i-1} - J_{i-1})$.

Theorem 2. *Using the initial value* $(D_i - J_i + C_i + B_i)/2$, *the recurrence relation converges to* R_i^{UB}, *where* $R_i \leq R_i^{UB} \leq D_i - J_i$, *if* τ_i *is schedulable.*

$$R_i^{LB} = \frac{C_i}{1 - \sum_{j=1}^{i-1} U_j} \tag{2}$$

Also alternative implementations were shown in the mentioned paper, resulting in a higher efficiency. Some of these will later be selected in the evaluation section of this paper.

Beside the previously mentioned task model, other task models may consist of tasks that have *static or dynamic offsets*, building so-called *linear or tree-shaped transactions*. This means that tasks are not activated independently from each other, but instead a group of tasks belonging to one transaction is activated with relative time offsets, forming a defined sequence of task activations. The first algorithm considering transactions was presented in (Tindell, 1994). In this approach, offsets are assumed to be static and smaller than the period of the corresponding transaction. It describes a technique to calculate the exact response times, which is computationally intractable for large task sets. Because the algorithm is NP-complete, the runtime grows exponentially depending on the number of tasks. This work was extended in (Palencia and Harbour, 1998) in the way that also dynamic offsets that may be greater than the period of the corresponding transaction are supported, as well as tasks that suspend themselves. The algorithm is called *WCDO (Worst Case analysis for Dynamic Offsets)*. Another exact schedulability test for tasks with static offsets was shown in (Lauer et al., 2010). In spite of the also exponential runtime, the evaluation results are better due to the assumption that there exist a single clock and no jitters in a distributed system. In this way, many cases can be excluded that have to be analyzed otherwise. Further work in this area has been done by (Palencia et al., 1999) and (Redell, 2004) by considering precedence relations of tasks. (Palencia et al., 1999) considers linear transactions, in which at most one task may depend directly on another. The presented algorithm is called *WCDOPS (Worst Case Dynamic Offsets with Priority Schemes)*. In (Redell, 2004) an improved *WCDOPS+* algorithm is presented, which additionally considers situations in which multiple tasks may depend on one task directly and are activated synchronously at the finishing time of this task. Such dependencies are called *tree-shaped transactions*. However, task sets with offsets are not further considered in this paper.

In this work, we focus on task sets without offsets in order to provide more efficient solutions for acceptance tests for these task sets. In the scope of this work we denote solutions efficient w.r.t. computational effort for analyzing schedulability for a task set. For task sets without offsets, it can be concluded that the improved variants of the RTA, using increased initial values and alternative implementation techniques, promise the best efficiency when applied as on-line acceptance test for task sets with arbitrary fixed priorities. However, the RTA can be further adapted to exploit this special use case. Also combinations of the RTA and the sufficient tests promise an increase of the overall performance. The developed adaptations of the RTA and also of some sufficient tests will be presented in the next section.

3 EFFICIENT ON-LINE ACCEPTANCE TESTS

A core requirement for on-line acceptance tests is efficiency w.r.t. computational effort, because the execution of the concurrently running tasks should not be influenced by the acceptance test. In our use case of executing a schedulability test as on-line acceptance test for new arriving tasks, the performance of the original exact RTA algorithm can be improved exploiting this special use case. This is, because the former task set has already been proven to be schedulable, yielding some information about their WCRTs that can be reused. Moreover, the performance can be increased by the fact, that not all of the existing tasks have to be proven again, but only a subset. In the following sections, some adaptations to the RTA and

also some promising sufficient (pessimistic) schedulability test algorithms are presented.

3.1 Adaptations of the RTA

In addition to using the improved initial values that were introduced in (Davis et al., 2008), further adaptations of the RTA are possible that exploit the special use case of an acceptance test, whereby the efficiency can be increased further.

While using the RTA as an acceptance test for a new task τ_{new}, the fact can be exploited that the former existent task set was schedulable and only the effects of τ_{new} have to be investigated. If this new task does not require any resources, it does not increase the maximum blocking times B_i of higher prioritized tasks. Hence, only the other tasks with equal or lower priority than τ_{new} have to be analyzed again. The higher prioritized tasks are not influenced by τ_{new} in this case, because their WCRT does not change. In the other case, in which τ_{new} increases the blocking time B_j of a higher prioritized task τ_j, even τ_j has to be tested again as well as all intermediate tasks with lower or equal priority than τ_j.

Because the former task set was analyzed in advance, we have already some knowledge about the WCRTs of the single tasks in this task set, due to the formerly executed RTA. These already computed response times can be reused later as initial values during the repeated execution of the acceptance test caused by a newly arriving task. Hence, there are two possibilities at this point. We could either use the improved initial values from (Davis et al., 2008)[1], or simply reuse the calculated old WCRTs as initial values for the anew analysis of the old tasks.

These old exact response times are valid initial values, because in single-core systems the new task may only change these response times to appear later, not earlier. Hence, the RTA recurrence relation will converge definitely and stay exact. However, the new task has to be analyzed with the initial value $C_{new} + B_{new}$ in the latter case, because no reusable response time is known. If a task leaves an execution platform, the former calculated WCRTs of the other tasks on this platform become pessimistic and should be recalculated in the idle time, before a new task arrives. Reusing the old WCRTs as initial values could be more efficient than calculating the initial values from (Davis et al., 2008), because nearly no computational overhead is required. Instead, the former stored values can be read and reused immediately. The effect of the chosen initial value is evaluated later in Section 4.4.

[1]Maximum of theorems 1, 2 and Equation (2).

An implementation of the RTA, which is usable as an efficient acceptance test for a new task τ_{new}, is shown as pseudo code in Figure 1. Only those tasks are analyzed again, which have a priority not greater than the highest prioritized task, whose blocking time is increased by τ_{new}. In the best case, τ_{new} produces no new blocking times, meaning that only tasks with $P_j \leq P_{new}$ have to be tested. In the worst case, the maximum blocking time of the highest prioritized task changes, necessitating a reanalysis of all tasks. It is assumed that for every existent task τ_j, there is a variable R_j^{old}, in which the calculated response time of a former executed RTA can be stored. Furthermore, it is assumed that tasks are ordered descending by their priorities and have unique id's from 1 to n, meaning that τ_1 has the highest priority and τ_2 has either the same or a lower one. Notice that this implementation is based on the alternative incremental implementation of the RTA introduced in (Davis et al., 2008).

The function *addToTaskSet* in line 2 adds τ_{new} to Γ_n and returns the merged task set Γ_{n+1}. The function *recalculateBlockingTimes* recalculates all blocking times B_i and returns the highest priority P_i of a task, for which B_i is increased by the new task τ_{new}. A higher value for P_i denotes a higher priority. The function *getInitialValue* in line 17 could return the mentioned old response time R_{ua}^{old} of the analyzed task τ_{ua}, or simply $C_{ua} + B_{ua}$, if the new arriving task is analyzed. Alternatively, the function could also return the maximum of theorems 1, 2 and Equation (2) for all tasks.

The incremental implementation is recognizable in lines 34-36. Here, R is not used as a constant during one iteration for all interfering tasks, but instead R is increased dynamically for each task. Afterwards, the increased R is used for the calculation of the interference by the next examined task, resulting in less required iterations and an increased performance.

This algorithm terminates immediately in line 8 as soon as one task has been found, which violates its deadline. Hence, the remaining tasks do not need to be tested, because the task set is already known to be not schedulable. Furthermore, the loops in line 19 and 31 have to iterate over all tasks, instead of running only up to the predecessor of τ_{ua}. This is necessary, because some equally prioritized tasks may be behind τ_{ua} in the ordered task list. Once a task with a lower priority has been reached, the loops can be aborted.

In spite of all these improvements, the computational effort of the exact RTA is still higher than for some sufficient but not necessary schedulability tests. Hence, under efficiency requirements, it would be helpful to use such a fast sufficient test in advance.

```
1:  procedure ACCEPTANCETEST(Taskset Γ_n, Task τ_new)
2:      Γ_{n+1} ← addToTaskSet(Γ_n, τ_new)
3:      P^high ← recalculateBlockingTimes(Γ_n, τ_new)
4:      for i ← 1 to n+1 do
5:          if P_i ≤ P^high then
6:              R_i ← getResponseTime(Γ_{n+1}, τ_i)
7:              if R_i > D_i − J_i then
8:                  return FALSE
9:              end if
10:         end if
11:     end for
12:     return TRUE
13: end procedure
14:
15: procedure GETRESPONSETIME(Taskset Γ_n, Task τ_ua)
16:     interference[n]
17:     R^prev ← getInitialValue(τ_ua)
18:     R ← C_ua + B_ua
19:     for j ← 1 to n do
20:         if P_j ≥ P_ua then
21:             if τ_j ≠ τ_ua then
22:                 interference[j] ← ⌈(R^prev + J_j)/T_j⌉ C_j
23:                 R ← R + interference[j]
24:             end if
25:         else
26:             break          ▷ all other tasks have P_j < P_ua
27:         end if
28:     end for
29:     while ((R > R^prev) and (R ≤ D_ua − J_ua)) do
30:         R^prev ← R;
31:         for j ← 1 to n do
32:             if P_j ≥ P_ua then
33:                 if τ_j ≠ τ_ua then
34:                     tmp ← ⌈(R+J_j)/T_j⌉ C_j
35:                     R ← R + (tmp - interference[j])
36:                     interference[j] ← tmp
37:                 end if
38:             else
39:                 break      ▷ all other tasks have P_j < P_ua
40:             end if
41:         end for
42:     end while
43:     return R
44: end procedure
```

Figure 1: Adapted implementation of the RTA for on-line acceptance tests.

Afterwards, the RTA has to be executed only in cases where the task set has not been identified as schedulable by the sufficient test before. One approach to provide a sufficient test is adapted in the following, in order to support also tasks with equal priorities, blocking times and release jitters.

3.2 Adaptations of Sufficient Tests

In this section, an adaptation of the sufficient test introduced in (Bini and Baruah, 2007) is provided. This adaptation extends the original test to support also blocking times B_i and release jitters J_i. The test is based on the calculations of upper bounds R_i^{UB} for the response time R_i of every task τ_i. If $\forall i : R_i^{UB} \leq D_i - J_i$, the task set is schedulable. If not, the schedulability is not known, because it may happen that $R_i \leq D_i - J_i < R_i^{UB}$.

The original equation from (Bini and Baruah, 2007) without B_i and J_i is now shown for comparison:

$$R_i^{UB} = \frac{C_i + \sum_{j \in hep(i)} C_j(1-U_j)}{1 - \sum_{j \in hep(i)} U_j}$$

This can be extended by B_i and J_i as follows:

$$R_i^{UB} = \frac{B_i + C_i + \sum_{j \in hep(i)} C_j(1-U_j) + \sum_{j \in hep(i)} J_j U_j}{1 - \sum_{j \in hep(i)} U_j} \tag{3}$$

To prove the correctness of Equation (3), let's consider without loss of generality a task τ_j with the exemplary parameters $J_j = 2, C_j = 3, T_j = 7$. What is the maximum workload $w_j(t)$ of this task in a time interval $[0,t]$? In the presence of release jitters, this is

$$w_j(t) = min\left\{t; (t+J_j) - (T_j - C_j)\left\lfloor \frac{t+J_j}{T_j} \right\rfloor; \left\lceil \frac{t+J_j}{T_j} \right\rceil C_j\right\}$$

In (Bini and Baruah, 2007), this workload function was similarly introduced, but without the J_j and the first candidate t, which is also required due to the release jitters. The way to achieve this workload function is visualized in Figure 2.

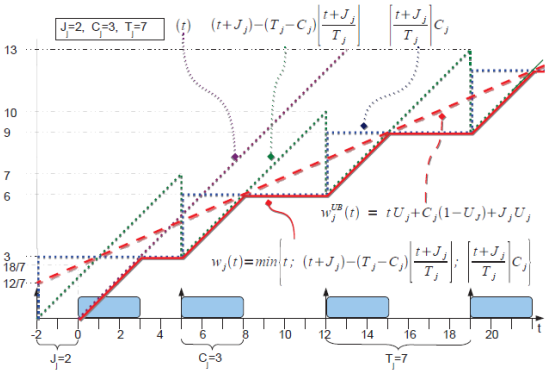


Figure 2: Finding a tight R^{UB} for a sufficient test.

In this figure, the three candidates for the minimum values of $w_j(t)$ are shown for the mentioned example task in the three dotted lines. The minimum of these candidates is denoted by the continuous line. To find an upper bound on $w_j(t)$, we have to

find a function that is never smaller than this continuous line. This is denoted by the dashed straight line. Like shown in (Bini and Baruah, 2007), the slope of this dashed line is tU_j. However, the value at time $t = 0$ changes in our case due to the release jitter to $C_j(1-U_j)+J_jU_j$, which is $18/7$ for the example task. Notice that the first instance of this task is activated at $t = -2$ and released at $t = 0$. In the absence of release jitters, the task would be released at $t = -2$, resulting in $C_j(1-U_j) = 12/7$. In conclusion, the function to calculate an upper bound on $w_j(t)$ can be expressed by:

$$w_j^{UB}(t) = tU_j + C_j(1-U_j) + J_jU_j$$

This results in the following upper bound R_i^{UB} for the worst case response time R_i.

$$\begin{aligned} R_i &\leq B_i + C_i + \sum_{j \in hep(i)} w_j^{UB}(R_i) \\ &= B_i + C_i + \sum_{j \in hep(i)} (R_iU_j + C_j(1-U_j) + J_jU_j) \\ R_i &\leq \frac{B_i + C_i + \sum\limits_{j \in hep(i)} C_j(1-U_j) + \sum\limits_{j \in hep(i)} J_jU_j}{1 - \sum\limits_{j \in hep(i)} U_j} = R_i^{UB} \end{aligned}$$

This is equal to Equation (3) and hence, the proof is completed.

Using this upper bound provides a tight sufficient test for task sets with arbitrary fixed priorities. However, calculating this upper bound for every analyzed task may require some computational effort due to the three sums in Equation (3). But as long as the tasks are analyzed in descending priority order, one possibility to confine this effort is to store the already computed sub-sums during the analysis of higher tasks, and to reuse them for analyzing the lower tasks. Thereby, the entire sums do not have to be computed newly for every analyzed task. Instead, the sums can simply be reused and extended during the sequence of analyzed tasks. Nevertheless, tasks with equal priorities must be considered explicitly, because all tasks with equal priorities to the analyzed task have to be treated like higher tasks. However, some of them may be ordered behind the analyzed task in the task list, which complicates the reuse of the sub-sums.

An efficient implementation of this sufficient test, which handles the problem of tasks with equal priorities and stores and reuses the sub-sums, is shown in Figure 3. The benefit of this response time based sufficient test compared to the utilization based sufficient test from (Liu and Layland, 1973) is that not only task sets with RM priority assignment are supported, but also task sets having arbitrary fixed priorities. Such task sets with arbitrary priorities may

```
 1: sumUj ← 0                                              ▷ ∑(U_j)
 2: sumCj1mUj ← 0                                          ▷ ∑(C_j(1−U_j))
 3: sumJjUj ← 0                                            ▷ ∑(J_jU_j)
 4: lastCheckedTaskPrio ← ∞;
 5:
 6: procedure SUFFICIENTTEST(Taskset Γ_n)
 7:     if ∑_{i=0}^{n−1} U_i > 1 then
 8:         return FALSE
 9:     end if
10:     for i ← 1 to n do
11:         R_i^UB ← getRUpperBound(Γ_n, τ_i)
12:         if R_i^UB > D_i − J_i then
13:             return FALSE
14:         end if
15:     end for
16:     return TRUE
17: end procedure
18:
19: procedure GETRUPPERBOUND(Taskset Γ_n, Task τ_ua)
20:     for j ← 1 to n do
21:         if lastCheckedTaskPrio > P_j and P_j ≥ P_ua then
22:             U_j ← C_j/T_j
23:             sumUj ← sumUj + U_j
24:             sumCj1mUj ← sumCj1mUj + C_j(1−U_j)
25:             sumJjUj ← sumJjUj + J_jU_j
26:         else if P_j < P_ua then
27:             break          ▷ all other tasks have P_j < P_ua
28:         end if
29:     end for
30:             ▷ remove the portion of τ_ua from the sums
31:     U_ua ← C_ua/T_ua
32:     sumUj_ua ← sumUj - U_ua
33:     sumCj1mUj_ua ← sumCj1mUj - C_ua(1−U_ua)
34:     sumJjUj_ua ← sumJjUj - J_uaU_ua
35:             ▷ calculate the final upper bound on R
36:     temp ← (B_ua+C_ua+sumCj1mUj_ua+sumJjUj_ua)
37:     R_ua^UB ← temp/(1−sumUj_ua)
38:     lastCheckedTaskPrio ← P_ua
39:     return R_ua^UB
40: end procedure
```

Figure 3: Implementation of the adapted sufficient test.

appear in different industrial application domains of embedded systems (e.g. the automotive domain) due to certain constraints.

4 EXPERIMENTAL RESULTS

In this section, we present evaluations of the previously introduced algorithms. The experiments are performed for task sets with rate monotonic (RM) priorities as well as for task sets with arbitrary fixed priorities. The algorithms are implemented in Java and executed on a standard personal computer, having a 3.16 GHz CPU. To reduce the impacts of the JIT-Compiler and Garbage-Collector to the measured time, the algorithms are executed multiple times per

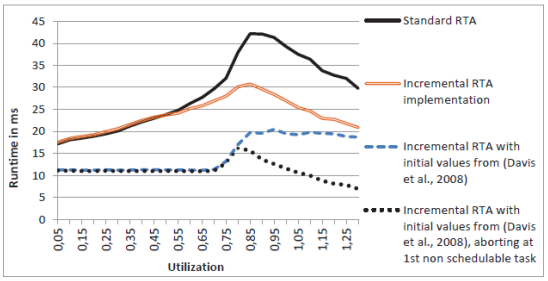
task set and the average runtime is determined. Furthermore, the algorithms are executed for 100 different task sets, which are generated randomly based on uniformly distributed task utilizations U_i. Each task set consists of 150 tasks. The random generation of the utilizations is based on the so-called *UUniFast* algorithm (Bini and Buttazzo, 2005). The other task parameters are either also determined randomly, or depending on other parameters (e.g. the WCET $C = U \cdot T$). However, some parameters have assumed restrictions. For RM tasks, B_i and J_i are set to zero and $D_i = T_i$. For arbitrary fixed priority task sets, B_i is random but at most equal to C_i, whereas J_i is at most 10% of T_i. Additionally, $D_i \leq T_i$ but at least 80% of T_i. Beside this, all tasks are independent, preemptive and have no offsets.

The remainder of this section presents the results of the experiments with the adapted schedulability tests (cf. sections 3.1 and 3.2) by means of their required execution time in *ms*. Each experiment is done for both types of randomly generated task sets, having either RM or arbitrary fixed priorities.

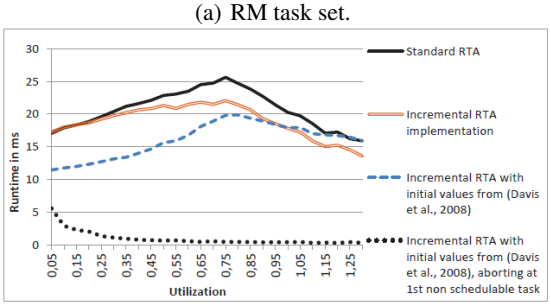
4.1 Improvements of the RTA

Figure 4 shows the evaluation results of the RTA improvements. This includes the standard RTA implementation (cf. Equation 1), its incremental implementation using standard initial values (algorithm from Figure 1, but not aborting in line 8 and *getInitialValue*(τ_{ua}) returns $B_{ua} + C_{ua}$) and the incremental implementation using the maximum of theorems 1, 2 and Equation (2) as initial value. These three algorithms are implemented in such a way that they do not terminate directly at the first found non-schedulable task. Instead, they continue and determine the total number of non-schedulable tasks in the task set. Though, this is not required for an on-line acceptance test. Because of this, the fourth (dotted) line in the figures shows the runtime of the incremental RTA using the maximum of the three mentioned initial value candidates that terminates immediately at the first found non-schedulable task (algorithm from Figure 1). Obviously, this is faster than continuing and checking also the remaining tasks.

Figure 4(a) shows the runtimes of the mentioned algorithms, if applied to task sets with RM priorities. Figure 4(b) shows the runtimes of the same algorithms, if applied to task sets with priorities assigned in an arbitrary manner. Because assigning priorities in a RM manner is optimal, the second priority assignment can only result in a lower degree of schedulability. It can be seen that the relative difference of the first three algorithms does not change significantly



(a) RM task set.



(b) Task set with arbitrary fixed priorities.

Figure 4: Using RTA for schedulability tests.

due to the used task model. The dashed line shows the incremental implementation using the maximum of theorems 1, 2 and Equation (2) as initial value and checking all tasks. This is faster than using the standard initial value in the standard and incremental RTA, shown by the first two lines.

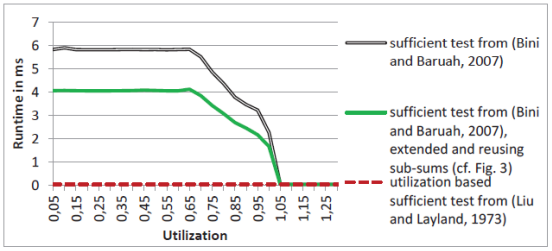
The main difference between the two task models appears at the fourth algorithm (dotted line) that immediately terminates at the first non-schedulable task. For RM task sets, every utilization up to around 69% is schedulable, because $U \leq n \cdot \left(2^{\frac{1}{n}} - 1\right) \leq 150 \cdot \left(2^{\frac{1}{150}} - 1\right) \approx 0.6948$. Hence, the speedup effect of this algorithm is only noticable for higher utilizations. However, in task sets with arbitrary priorities, even very low utilizations may be not schedulable. This results in the much lower runtime of the fourth algorithm in Figure 4(b), because non-schedulable tasks are found even for low utilizations.

4.2 Improvements of Sufficient Tests

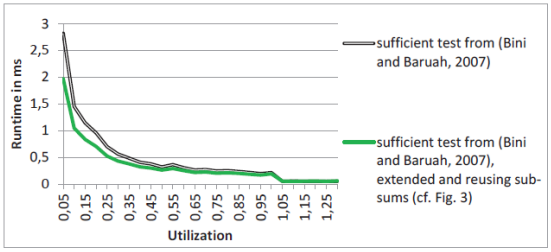
The results of applying sufficient tests to task sets with RM or arbitrary priorities are shown in figures 5(a) and 5(b). Both figures show the runtimes of the original response time based test from (Bini and Baruah, 2007) and its extended efficient implementation, introduced in Figure 3.

It can be seen in Figure 5(a) that for RM task sets the runtime of the implementation from Figure 3 is nearly 30% lower than the runtime of the original

variant. Notice that both runtimes begin to decrease for utilizations bigger than round about 69%. This is, because the test immediately terminates, as soon as a non-schedulable task is found. For lower utilizations, no non-schedulable tasks are found. Therefore, the runtime keeps constant. Also the utilization based test from (Liu and Layland, 1973) is shown in Figure 5(a). Its runtime is by far the best, but it cannot be applied for tasks with arbitrary priorities. Hence, this test is not shown in Figure 5(b). This figure shows that the runtime of both response time based sufficient tests is much lower than in the RM case. This occurs due to the existing non-schedulable tasks even for low utilizations. The implementation introduced in Figure 3 is again faster than the original version because of the reuse of already computed sub-sums.



(a) RM task set.



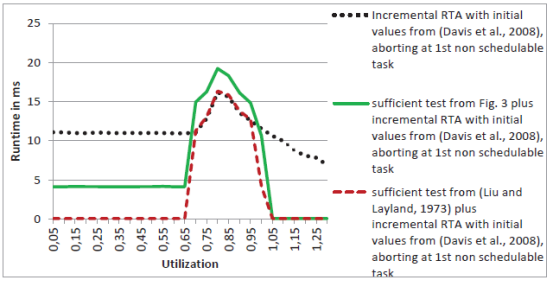
(b) Task set with arbitrary fixed priorities.

Figure 5: Experimental results of sufficient tests.

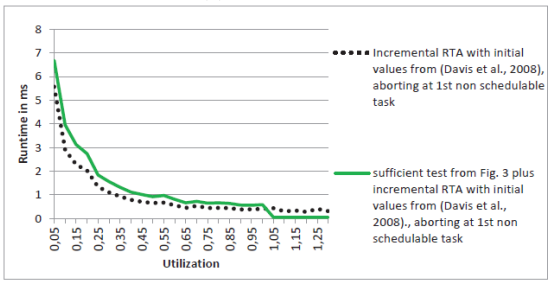
4.3 Using a Sufficient Test Prior to the RTA

The execution times of the most sufficient tests are much better than the runtimes of the RTA variants, evaluated in Section 4.1. However, they are pessimistic in some way and cannot determine all schedulable task sets. Hence, it is promising to combine both algorithm types. At first, a relatively fast sufficient test is applied, and afterwards the slower RTA is applied only if the sufficient test fails.

Figure 6(a) shows the results of this combination, when applied to RM task sets. The results of the combinations are shown as continuous and dashed line. The dotted line shows the runtime of solely executing the incremental RTA with improved initial values. For



(a) RM task set.



(b) Task set with arbitrary fixed priorities.

Figure 6: Experimental results of combinations of RTA and sufficient tests.

$U \leq 69\%$, the sufficient tests are always successful, redundantizing the RTA. Thus, both combinations are faster than using the single RTA in this area. The continuous line shows the runtime, if the sufficient test from Figure 3 is used prior to the RTA. The dashed line shows the same combination, but using the sufficient utilization based test introduced in (Liu and Layland, 1973) instead. Our evaluation shows that the latter combination is much faster than the former one. However, for utilizations higher than 69%, both sufficient tests cannot guarantee the schedulability in all cases. Hence, the RTA has to be executed subsequently, too. This results in an overall runtime which is higher than the dotted line. Because a task set cannot be schedulable for $U > 1$, the RTA is not executed subsequently in this case, resulting in a runtime near to zero.

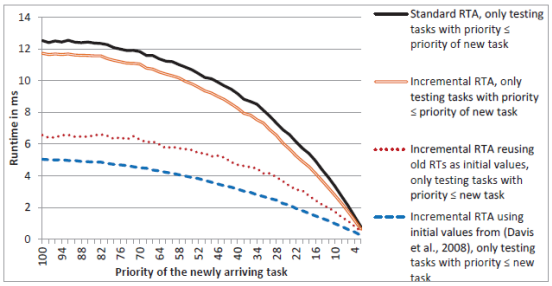
Figure 6(b) shows the analogous experiment applied for arbitrary fixed priority task sets. Only the sufficient test presented in Figure 3 is used in combination with the RTA here, because the other sufficient test from (Liu and Layland, 1973) is not applicable. Because the sufficient test fails in almost all cases even for low utilizations, no efficiency improvement can be achieved and the total runtime of the combination is greater than using the RTA alone.

4.4 Adding a New Task

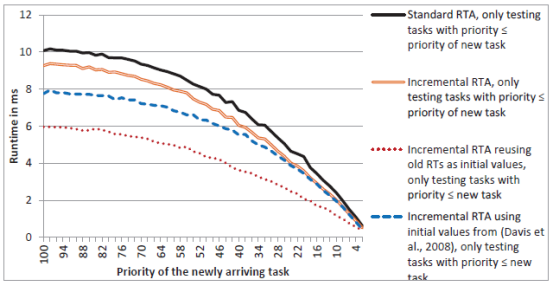
As mentioned before, the exploitation of the follow-

ing situation can also lead to a further improvement of the RTA: In the case of an arrival of a new task, only those tasks have to be tested again, which are influenced by this new task directly via preemption or via increased blocking times. This is valid under the assumption that the task set was schedulable before, without the new task. An implementation of this approach is presented in Figure 1.

Thus, if the RTA is used as an acceptance test for a new task arriving in the system and if only tasks with priorities lower or equal to the new task are tested again, this results in the runtimes that are shown in Figure 7.



(a) RM task set.



(b) Task set with arbitrary fixed priorities.

Figure 7: Experimental results of RTA variants, used as acceptance test for one new added task.

All evaluations are performed under the assumptions that the already existing task set comprises 100 tasks with priorities from 1 to 100, and that the total utilization of the existing task set is 70%. Then, one new task with variable priority between 1 and 100 is added, wherefore the acceptance test is executed. It is assumed that the new task does not increase any blocking times of higher prioritized tasks. Hence, no higher prioritized task has to be tested. If otherwise the new task would increase any blocking times of such tasks and these tasks must also be tested again. This would result in a lower performance.

It can be seen in both figures that the lower the priority of the new task is, the less tasks have to be tested again and the less the computational effort is. The curves are convex, because analyzing a task with low priority requires more time than analyzing a task with high priority. Hence, the speedup effect per omitted analysis of a low priority task is higher than per omitted high priority task.

As before, the standard RTA implementation has the lowest performance (upper continuous line) and is improved by its incremental implementation (lower continuous line). This is further improved again, if we use better initial values. It can be seen in Figure 7(a) that for RM task sets, it is better to use the initial values from (Davis et al., 2008) (dashed line) than using the old response times as initial values (dotted line). This is surprising in such a way that the calculation of the sophisticated initial values requires some amount of time, while reusing the old response times generates no overhead. But the initial values from (Davis et al., 2008) are much better, so that the overhead of their calculation can be compensated. The RM task set has unique RM priorities between 1 and 100 in this experiment. Thereby, the periods are unique, too.

Figure 7(b) shows the analogous situation, but the existing task set has arbitrary fixed priorities, which are not assigned in a RM manner. Therefore, the priorities and periods are independent, but both in a range from 1 to 100. In contrast to the results presented in Figure 7(a) for the RM case, the approach which uses the old response times of the existing task set as initial values is the most efficient one. The reason for this is that the sophisticated initial values from (Davis et al., 2008) are almost always bigger than the old response times for RM task sets. For task sets with arbitrary fixed priorities, this is not the case. Hence, the overhead for calculating the sophisticated initial values cannot be compensated, here.

Though, due to the randomly generated task sets, it is not ensured during the evaluation in Figure 7 that the old task set was schedulable before. Because of this, only variants of algorithms are evaluated that do not terminate at the first non-schedulable task, but also analyze the remaining tasks. This allows a useful statement about their efficiency, regardless whether the old task set was schedulable before, or not. But obviously, if we do not analyze the whole task set, the result of the analysis cannot be secure in such cases where the old task set was not schedulable, because some non-schedulable tasks may not be tested and hence not found.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we analyzed some existing schedulability tests according to their applicability as efficient

on-line acceptance test in adaptive networked embedded real-time systems. The aim is to enable runtime adaptation in networked embedded systems with hard real-time requirements. Some initially selected schedulability tests were adapted in order to increase their efficiency in the use case of an on-line acceptance test for new arriving tasks, or to make them in general applicable to our considered task sets. This includes the response time analysis and an existing sufficient test. Also efficient implementations of both algorithms were shown.

To estimate the benefits of the provided adaptations, several types of experiments were done. Also combinations of algorithms were evaluated, in which sufficient tests are executed primary and the RTA which is only executed in unsuccessful cases. While for RM task sets, such combinations result in a huge overall performance increase (cf. Figure 6(a)), this effect vanishes for task sets with arbitrary priorities (cf. Figure 6(b)). Furthermore, we evaluated the use case of adding a new task to an existing task set. For RM task sets, using the initial values from (Davis et al., 2008) was the fastest solution (cf. Figure 7(a)), but for tasks with arbitrary fixed priorities simply reusing the old response times as initial values was the fastest approach (cf. Figure 7(b)). However, with both initial values the adapted RTA results in a far better average runtime, compared to analyzing all tasks every time.

In future work, the results achieved in this paper for task sets without offsets are intended to be extended to offset based task sets. This includes on-line acceptance tests for task sets with static or dynamic offsets in linear or tree-shaped transactions. Also an analysis should be done about combined task sets, comprising tasks without offsets as well as tasks in transactions.

REFERENCES

Audsley, N., Burns, A., Richardson, M., Tindell, K., and Wellings, A. (1993). Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, pages 284–292.

Audsley, N., Burns, A., Richardson, M., and Wellings, A. (1991). Hard real-time scheduling: the deadline-monotonic approach. In *Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems and Software*, pages 133–137.

Bini, E. and Baruah, S. (2007). Efficient computation of response time bounds under fixed-priority scheduling. *Proceedings of the 15th Int. Conference on Real-Time and Network Systems*, pages 95–104.

Bini, E. and Buttazzo, G. (2005). Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1):129–154.

Davis, R., Zabos, A., and Burns, A. (2008). Efficient exact schedulability tests for fixed priority real-time systems. *IEEE Transactions on Computers*, 57(9):1261–1276.

Joseph, M. and Pandya, P. (1986). Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395.

Lauer, C., Hielscher, K., German, R., and Pollmer, J. (2010). Schedulability Analysis in Time-Triggered Automotive Real-Time Systems. In *IEEE Vehicular Technology Conference Fall*, pages 1–5.

Lei, W., Zhaohui, W., and Mingde, Z. (2004). Worst-case response time analysis for OSEK/VDX compliant real-time distributed control systems. In *Proceedings of the 28th Int. Computer Software and Applications Conference*, pages 148–153.

Liu, C. and Layland, J. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61.

McKinley, P., Sadjadi, S., Kasten, E., and Cheng, B. (2004). Composing adaptive software. *IEEE Computer*, 37(7):56–64.

Oreizy, P., Medvidovic, N., and Taylor, R. N. (1998). Architecture-based runtime software evolution. In *Proceedings of the 20th International Conference on Software Engineering*, pages 177–86.

Palencia, J. and Harbour, M. (1998). Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pages 26–37.

Palencia, J., Harbour, M., et al. (1999). Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, pages 328–339.

Pop, T., Eles, P., and Peng, Z. (2003). Schedulability analysis for distributed heterogeneous time/event-triggered real-time systems. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, pages 257–266.

Redell, O. (2004). Analysis of tree-shaped transactions in distributed real time systems. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, pages 239–248.

Streichert, T., Haubelt, C., Koch, D., and Teich, J. (2008). Concepts for self-adaptive and self-healing networked embedded systems. *Organic Computing*, pages 241–260.

Tindell, K. (1994). Adding time-offsets to schedulability analysis. *Department of Computer Science, University of York, Report Number YCS-94-221*.

Weiss, G., Zeller, M., and Eilers, D. (2011). Towards automotive embedded systems with self-x properties. In *New Trends and Developments in Automotive System Engineering*. InTech.

Zeller, M., Weiss, G., Eilers, D., and Knorr, R. (2011). An approach for providing dependable self-adaptation in distributed embedded systems. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 236–237.