Response Time Upper Bounds for Fixed Priority Real-Time Systems.

R.I.Davis and A.Burns
Real-Time Systems Research Group, Department of Computer Science,
University of York, YO10 5DD, York (UK)
rob.davis@cs.york.ac.uk, alan.burns@cs.york.ac.uk

Abstract

This paper derives closed form upper bounds on the response times of tasks in fixed priority real-time systems. These bounds are valid for tasks with arbitrary deadlines, release jitter, and blocking. Response time upper bounds are given for tasks that are scheduled pre-emptively, co-operatively with intervals where pre-emption is deferred, and non-preemptively. The set of upper bounds for n tasks can be computed in O(n) time, providing a linear-time sufficient schedulability test, applicable to complex commercial real-time systems.

1. Introduction

1.1. Background and motivation

Fixed priority scheduling is widely used in real-time embedded systems, such as electronic control units and communications networks in automobiles, digital set-top boxes, medical systems, space systems, and mobile phones. As a result, fixed priority scheduling is supported by the majority of commercial real-time operating systems.

In the context of fixed priority systems, schedulability analysis is used to determine if a set of tasks or messages can be guaranteed to always meet their deadlines at run-time. A common method of determining schedulability is to calculate the worst-case *response time* of each task, measured from its release to its completion, and then compare this response time with the task's deadline to determine if it is schedulable. This method is called *Response Time Analysis* (RTA).

Exact response time analysis is known to have pseudo-polynomial complexity [1, 14]; however, in practice, efficient implementations exist [5, 8, 24] that can be used to determine the schedulability of industrial-scale real-time systems within a reasonable time frame. Nevertheless, there are a number of ways in which schedulability tests can be used, where the complexity and execution time of exact tests may become a limiting factor. These include:

1. Interactive system design tools: Here user interaction / sensitivity analysis [20] requires that the results of multiple schedulability tests, on large and complex systems, are available within a few tenths of a second.
2. System optimisation via search: Using search techniques such as simulated annealing to determine task and message allocations in a distributed system [9] requires a very large number of schedulability tests to be performed.
3. Dynamic systems: In a running system, admission of a new task requires an online assessment of system schedulability before the task can be started. Stringent start-up constraints may preclude the use of exact schedulability tests in this context.

In 2008, Davis et al. [8] showed that the efficiency of exact tests can be markedly improved, by using a response time upper bound to check on a task-by-task basis whether an exact response time calculation is required. Reducing the execution times of exact tests in this way can significantly enhance their suitability for use in practical applications such as those described above.

1.2. Related work

Research into schedulability tests for fixed priority pre-emptive systems effectively began in 1967, when Fineberg and Serlin [10] considered priority assignment for two tasks. They noted that if the task with the shorter period is assigned the higher priority, then the least upper bound on the schedulable utilisation is $2(\sqrt{2}-1)$ or 82.8%. This result was generalised by both Serlin [23] in 1972 and Liu and Layland [19] in 1973, both of whom showed that for *synchronous* tasks (that share a common release time), that comply with a restrictive system model, and that have deadlines equal to their periods, then *rate monotonic* priority ordering is optimal. Liu and Layland [23] provided a simple, sufficient, utilisation-based schedulability test for tasks compliant with their system model, and with priorities in rate monotonic priority order (RMPO)[1].

In 2003, Lauzac et al [15] introduced an improved utilisation-based test, using period ratios, called the RBound test. Also in 2003, Bini et al. [4] introduced a utilisation-based test referred to as the Hyperbolic

[1] RMPO assigns priorities in order of task periods, such that the task with shortest period is given the highest priority.

Bound. Both the RBound and Hyperbolic bound are only applicable to systems with a restricted system model, and task priorities assigned in RMPO[2].

Exact schedulability tests, based on computing worst-case response times, were introduced by Joseph and Pandya [14] in 1986, and Audsley et al. [1] in 1993. An alternative method of determining exact schedulability without computing worst-case response times was introduced by Lehoczky et al. [17] in 1989.

These exact tests have been extended to cater for cases where tasks access mutually exclusive shared resources according to mechanisms such as the Stack Resource Policy [2] and the Priority Ceiling Protocol (PCP) [22]. Further work on schedulability tests for fixed priority systems has lifted many of the earlier restrictions, providing exact tests for tasks with offset release times [25], arbitrary deadlines [16, 26], release jitter [26] and non-pre-emptive sections [6, 13]. Exact schedulability tests have also been developed for other resources scheduled according to fixed priorities, for example Controller Area Network [7].

Research by Sjodin and Hansson [24] in 1998, Bril et al. [5] in 2003, and Davis et al. [8] in 2008, has lead to practical improvements in the efficiency of exact response time tests, providing response time lower bounds that can be used as initial values for the fixed point equation used in such tests. Nevertheless the exact response time tests remain pseudo-polynomial in complexity.

A polynomial-time, sufficient response time test was developed by Fisher and Baruah [12] in 2005. This test approximates the workload requested by higher priority tasks using an exact request bound function for k invocations of the task, and a linear function thereafter. This work has subsequently been extended to cater for tasks with release jitter [21].

A closed form upper bound on task's response times, and an associated linear-time sufficient test was derived by Bini and Barauh [3] in 2007. This response time upper bound is valid for sets of independent, pre-emptively scheduled tasks with arbitrary deadlines, and no release jitter.

1.3. This paper

The research described in this paper was motivated by the need to provide response time upper bounds that can be used as an integral part of exact response time analysis for complex commercial real-time systems.

The research presented in the rest of this paper builds upon the work of Bini and Baruah [3]. It takes the concept of approximating task workload via a linear function and applies it to systems where tasks may be subject to blocking, have arbitrary deadlines, and arbitrary release jitter. Further, in this paper, response time upper bounds are derived for pre-emptive, co-operative, and non-pre-emptive scheduling, ensuring that the results are applicable to a wide range of processor and network scheduling problems.

[2] Later in this paper, we show how these utilisation-based tests can be adapted to more complex system models.

1.4. Organisation

Section 2 describes the terminology, notation and system models used in the rest of the paper. Section 3 summarises existing response time analysis. Section 4 derives response time upper bounds for fixed priority tasks with arbitrary deadlines, release jitter, and blocking, under pre-emptive, co-operative, and non-pre-emptive scheduling. These results form a sufficient schedulability test that is applicable to a very general system model. Section 5 presents an empirical investigation into the effectiveness of the sufficient test / response time upper bound. Finally, Section 6 summarises the key contributions of this paper and suggests directions for future research.

2. System model, terminology and notation

In this paper, we are interested in providing a closed form response time upper bound that can be used to form a linear-time sufficient test for applications executing under a fixed priority scheduler on a single processor.

The application is assumed to comprise a static set of n tasks ($\tau_1..\tau_n$), each assigned a unique priority i, from 1 to n (where n is the lowest priority).

We use the notation *hp(i)* and *lp(i)* to mean the set of tasks with priorities higher than i, and the set of tasks with priorities lower than i respectively. Similarly, we use the notation *hep(i)* and *lep(i)* to mean the set of tasks with priorities higher than or equal to i, and lower than or equal to i respectively.

Application tasks may arrive either *periodically* at fixed intervals of time, or *sporadically* after some minimum inter-arrival time has elapsed. Each task τ_i, is characterised by: its relative *deadline* D_i, *worst-case execution time* C_i, minimum inter-arrival time or *period* T_i, and its *release jitter* J_i, defined as the maximum time between the task arriving and it being released (ready to execute). The *utilisation* U_i, of each task is given by C_i / T_i.

No assumptions are made about the relationship between the period of a task and its release jitter or between the period of a task and its deadline. Arbitrary release jitter ($0 \le J_i < D_i$), and arbitrary deadlines ($D_i \le T_i$ or $D_i > T_i$) are therefore permitted. It is assumed that invocations of a task are released, and execute in order of arrival, and that once a task starts to execute it will not suspend itself voluntarily.

We assume that the arrival times of tasks are independent and thus that the tasks may share a common release time, termed the *critical instant* [19]. Tasksets with a common release time are referred to as *synchronous* tasksets. Tasksets with offset arrival times

that do not share a common release time are referred to as *asynchronous* tasksets. Asynchronous tasksets are not explicitly considered in this paper; however, the response time upper bounds provided, although pessimistic in this case, are applicable to such tasksets.

Tasks may access shared resources in mutual exclusion according to the Stack Resource Policy [9]. A task at priority *i* may be blocked by lower priority tasks, as a result of the operation of the Stack Resource Policy, for at most B_i, referred to as the *blocking time*.

A task's *worst-case response time* R_i, is the longest time from the task's release to it completing execution. A task is referred to as *schedulable* if its worst-case response time is less than or equal to its deadline minus release jitter ($R_i \le D_i - J_i$). A system is referred to as schedulable if all its tasks are schedulable.

We assume, without loss of generality[3], that task priorities are in *deadline minus jitter monotonic*[4] (D-JMPO) priority order [27].

In this paper, we provide analysis for fully pre-emptive systems, and also for systems where pre-emption may be deferred [6], so called *co-operative* scheduling. With co-operative scheduling, it is assumed that each task's worst-case execution time C_i is divided into a number of non-pre-emptable sections, the final one of which is of length F_i ($F_i \le C_i$), and that the blocking time B_i represents the longest time for which any task of lower priority than *i* can execute non-pre-emptively. Note, that pure non-pre-emptive scheduling is a special case of deferred pre-emption, with $F_i = C_i$.

2.1. Busy, idle and occupied periods

A *priority level-i idle instant* is defined as a time instant *t* at which there are no tasks of priority *i* or higher awaiting execution that became ready to execute strictly before *t*.

A *priority level-i busy period* is defined as a time interval $[t_1, t_2)$ which; (i) starts at a priority level-*i* idle instant t_1, when a task of priority *i* or higher becomes ready to execute, (ii) is a contiguous interval of time during which any task of priority lower than *i* is unable to start executing, and (iii) ends at the first priority level-*i* idle instant t_2 after its start.

A *priority level-i idle period* is defined as a time interval $[t_3, t_4)$ of length greater than zero, during which no tasks are ready to execute at priority *i* or higher, strictly before the end of the idle period at t_4.

A *priority level-i occupied period* is defined as a time interval $[t_5, t_6)$ which; starts at the end of a priority level-*i* idle period and ends at the start of the next priority level-*i* idle period. During a priority level-*i* occupied period, the processor is occupied executing tasks at priority *i* or higher and is thus unable to start execution of any computation at a priority lower than *i*.

Note, the subtle difference between a *busy period* and an *occupied period*: a *busy period* ends once all tasks released strictly prior to the end of the *busy period* have completed execution, irrespective of whether further tasks are released at the end of the period; whereas, an *occupied period* only ends once all tasks released prior to, and including, the end of the period have completed execution.

Busy periods are fundamental in the analysis of fixed priority pre-emptive scheduling, whilst *occupied periods* are fundamental in analysing co-operative and non-pre-emptive scheduling.

In the remainder of the paper, for conciseness, we will use the terms *busy period* and *occupied period* whenever the priority level is implicit in the context. Further, when we refer to a busy period or occupied period, we mean the worst-case or longest such period.

2.2. Summary of notation used

For ease of reference, Table 1 below summarises the notation used throughout this paper.

Table 1: Notation

Symbol	Description
τ_i	Task at priority level *i*.
B_i	Blocking time at priority level *i*.
C_i	Worst-case execution time of task τ_i
D_i	Deadline of task τ_i
F_i	Length of the final non-pre-emptable section of task τ_i
$I_j^O(t)$	Interference at priority *j* executed by the processor in an interval of length *t*, when τ_j is the only task in the system.
$I_i^{UB}(t)$	Upper bound on $I_j^O(t)$.
J_i	Release jitter of task τ_i
$O_i(C)$	Worst-case priority level-*i* occupied time, including computation *C* at priority *i*.
$O_i^{UB}(C)$	Upper bound on $O_i(C)$
q	Denotes an invocation of task τ_i ($q = 0$ is the first invocation).
R_i	Worst-case response time of task τ_i
$R_i(q)$	Worst-case response time of the *q*th invocation of task τ_i.
$R_i^{UB}(q)$	Upper bound on $R_i(q)$.
R_i^{UB}	Upper bound on R_i.
T_i	Minimum inter-arrival time of task τ_i
U_i	Utilisation of task τ_i
v_i	Length of a priority level-*i* occupied period.
$v_i(q)$	Length of a priority level-*i* occupied period, up to the start of the final non-pre-emptable section of the *q*th invocation of task τ_i.
$V_i^{UB}(q)$	Upper bound on $v_i(q)$.
w_i	Length of a priority level-*i* busy period.
$w_i(q)$	Length of a priority level-*i* busy period, up to the end of the *q*th invocation of task τ_i.
$W_i^{UB}(q)$	Upper bound on $w_i(q)$.

[3] The analysis provided in this paper is independent of the chosen priority ordering.

[4] D-JMPO assigns priorities in order of deadline minus jitter, such that the task with the smallest value of D_i-J_i is given the highest priority.

3. Response time analysis

In this section, we outline standard response time analysis methods for fixed priority systems. The overall approach taken by response time analysis can be summarised as follows:

1. Determine the worst-case scenario; the pattern of task activation and execution that leads to task τ_i having its worst-case response time R_i.
2. Compute R_i for the worst-case scenario.
3. Determine task schedulability by checking whether $R_i \le D_i - J_i$.

For synchronous systems, Tindell [26] showed that the worst-case scenario for task τ_i occurs following a critical instant where τ_i is released simultaneously with all higher priority tasks, all subject to their maximum release jitter, and subsequent releases of task τ_i and higher priority tasks then occur after the minimum possible time intervals. Further, immediately prior to the critical instant, a lower priority task has just locked a shared resource or entered a non-pre-emptive section, and will therefore execute at priority level *i* or higher for the worst-case blocking time B_i.

The worst-case scenario described above leads to the longest priority level-*i* busy period and the longest priority level-*i* occupied period [13].

In the rest of this section, we provide an outline of the analysis used to compute worst-case response times:

(i) For pre-emptively scheduled tasks with deadlines less than or equal to their periods [1] (Section 3.1).

(ii) For pre-emptively scheduled tasks with arbitrary deadlines [26] (Section 3.2).

(iii) For pre-emptively scheduled tasks with deferred pre-emption [6] (Section 3.3).

3.1. Pre-emptive scheduling, with deadlines less than or equal to periods

For pre-emptively scheduled tasks with deadlines less than or equal to their periods, each invocation of task τ_i must be complete before the next invocation is released. Hence any priority level-*i* busy period contains at most one invocation of task τ_i. The worst case response time of task τ_i therefore equates to the length of the longest priority level-*i* busy period, starting at the critical instant

Response time analysis [1, 14], calculates the length w_i, of this busy period using the following equation.

$$w_i = B_i + C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil C_j \qquad (1)$$

where the summation term represents the total interference due to invocations of higher priority tasks released strictly before the end of the busy period.

Note that w_i appears on both the left and right hand sides of Equation (1). As the right hand side is a monotonically non-decreasing function of w_i, the equation can be solved using the following fixed point iteration:

$$w_i^{n+1} = B_i + C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i^n + J_j}{T_j} \right\rceil C_j \qquad (2)$$

Iteration starts with an initial value w_i^0, typically $w_i^0 = B_i + C_i$, and ends when either $w_i^{n+1} = w_i^n$ in which case the worst-case response time R_i, is given by w_i^{n+1} or when $w_i^{n+1} > D_i - J_i$ in which case the task is unschedulable. The fixed point iteration is guaranteed to converge provided that the overall taskset utilisation is less than 1.

3.2. Pre-emptive scheduling with arbitrary deadlines

For pre-emptively scheduled systems, where task deadlines are arbitrary, execution of one invocation of a task may not necessarily be complete before the next invocation is released. Hence a number of invocations of task τ_i may be present within the longest priority level-*i* busy period, with earlier invocations delaying the execution of later ones. In general it is therefore necessary to compute the response times of all invocations within the busy period in order to determine the worst-case response time.

We note that with arbitrary release jitter, a number of invocations of task τ_i may be released together at the start of the busy period; such invocations of the same task are assumed to be executed in order of arrival[5].

The length of the busy period $w_i(q)$, starting at the critical instant and extending until the completion of the *q*th invocation of τ_i (where $q = 0$ is the first invocation) is given by the fixed point iteration:

$$w_i^{n+1}(q) = B_i + (q+1)C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i^n(q) + J_j}{T_j} \right\rceil C_j \qquad (3)$$

Iteration starts with an initial value $w_i^0(q)$, typically $w_i^0(q) = B_i + (q+1)C_i$, and ends when either $w_i^{n+1}(q) = w_i^n(q)$ in which case the worst-case response time $R_i(q)$, of invocation *q*, is given by $w_i^{n+1}(q) - qT_i$ or when $w_i^{n+1}(q) - qT_i > D_i - J_i$ in which case invocation *q* is unschedulable.

Invocation *q* can only impinge upon the execution of subsequent invocations if its completion occurs after their release. Hence, response times need to be calculated for invocations *q*=0,1,2,3… until an invocation *q* is found that completes at or before the earliest possible release of the next invocation *q*+1, i.e. where: $w_i(q) \le (q+1)T_i - J_i$. The worst-case response time of task τ_i is then given by:

[5] We assume that a later arriving invocation cannot, by virtue of having less release jitter, overtake an earlier one and be released first.

$$R_i = \max_{\forall q}(w_i(q) - qT_i) \quad (4)$$

Again, the task is schedulable provided that $R_i \le D_i - J_i$.

3.3. Co-operative scheduling

In the case of co-operative scheduling, even if task deadlines are less than or equal to their periods, an invocation of task τ_i can still delaying the execution of later invocations. This happens because non-pre-emptive execution of the final section of task τ_i can delay execution of a higher priority task which then impinges upon the next invocation of task τ_i [6]. Response time analysis for pre-emptive scheduling with deferred pre-emption therefore needs to consider multiple invocations even when task deadlines are less than or equal to their periods.

In a co-operatively scheduled system, a pessimistic value [6] for the length of the priority level-*i* worst-case occupied period $v_i(q)$, by which invocation *q* of task τ_i may be delayed from starting its final non-pre-emptable section is given by the solution to the following fixed point equation:

$$v_i^{n+1}(q) = B_i + (q+1)C_i - F_i + \sum_{\forall j \in hp(i)} \left(\left\lfloor \frac{v_i^n(q) + J_j}{T_j} \right\rfloor + 1 \right) C_j \quad (5)$$

where the summation term represents the total interference due to invocations of higher priority tasks released at or before $v_i^n(q)$, and B_i is the maximum blocking time due to the non-pre-emptive execution of sections of lower priority tasks.

Equation (5) can be solved using a fixed point iteration, that starts with an initial value $v_i^0(q)$, typically $v_i^0(q) = B_i + (q+1)C_i - F_i$, and ends when either $v_i^{n+1}(q) = v_i^n(q)$ in which case the worst-case response time $R_i(q)$, of invocation *q*, is given by $v_i^{n+1}(q) + F_i - qT_i$ or when $v_i^{n+1}(q) + F_i - qT_i > D_i - J_i$ in which case the invocation is unschedulable.

The number of invocations which need to be examined is given by:

$$Q_i = \lceil (w_i + J_i)/T_i \rceil \quad (6)$$

Where w_i is the length of the priority level-*i* busy period assuming pre-emptive scheduling, found using the following fixed point iteration, starting with a value of $w_i^0 = B_i + C_i$ and ending when $w_i^{n+1} = w_i^n$.

$$w_i^{n+1} = B_i + \sum_{\forall j \in hep(i)} \left\lceil \frac{w_i^n + J_j}{T_j} \right\rceil C_j \quad (7)$$

The worst-case response time of task τ_i is then given by:

$$R_i = \max_{q=0,1..Q_i-1}(v_i(q) + F_i - qT_i) \quad (8)$$

Again, the task is schedulable provided that $R_i \le D_i - J_i$.

We note that purely non-pre-emptive scheduling [13] is a special case of pre-emptive scheduling with deferred pre-emption, with the final non-pre-emptive section equal to the entire execution of the task ($F_i = C_i$).

4. Response time upper bounds

In this section, we present response time upper bounds for fixed priority systems, with arbitrary deadlines and release jitter. Following the approach of Bini and Baruah [3], we first derive a linear function that provides a closed form upper bound on the time that the processor can spend executing a high priority task τ_j during any given time interval. We then show how this linear function can be used to provide an upper bound on the worst-case occupied time and hence to provide response time upper bounds for pre-emptive, co-operative, and non-pre-emptive scheduling.

4.1. Interference upper bound

In this section, we derive a linear function that forms an upper bound on the total time $I_j(t)$, that the processor spends executing a task τ_j during the interval [0, *t*) in the worst-case scenario (Section 2.1). We refer to $I_j(t)$ as the *interference* due to task τ_j.

Following the approach outlined in [3], let $I_j^O(t)$ denote the worst-case interference due to task τ_j, when it is the *only* task in the system. Clearly $I_j^O(t) \ge I_j(t)$.

Assuming that τ_j is the only task in the system, then with arbitrary release jitter J_j, a number of invocations of τ_j may be released simultaneously at time *t*=0 and executed consecutively. Further invocations of τ_j may also be released prior to these initial invocations completing execution. In general, the processor will first be busy executing *h* invocations of τ_j, then it will be idle for some time interval ($\le T_j$) before subsequently executing for C_j every T_j, as shown in Figure 1.

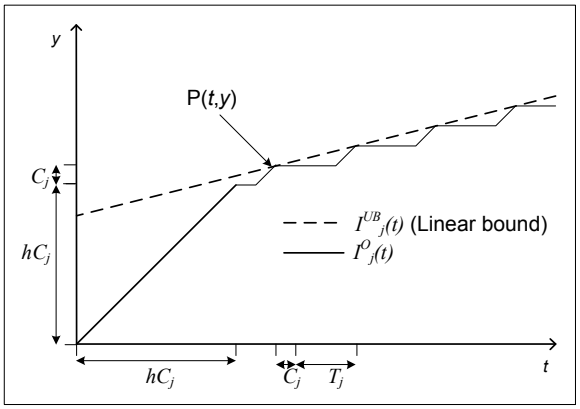

Figure 1

We now derive an equation for the linear upper bound $I_j^{UB}(t)$, on $I_j^O(t)$ shown as a dashed line in

Figure 1. This upper bound has a slope of $U_j = C_j / T_j$ and is the smallest linear function of t, with slope U_j, such that $\forall t \quad I_j^{UB}(t) \geq I_j^O(t)$. The point P($t$, y) in Figure 1 represents the point with the smallest value of t, for which $I_j^{UB}(t) = I_j^O(t)$.

To determine the equation for $I_j^{UB}(t)$, we must first compute the number of invocations of τ_j that execute consecutively, this information can then be used to determine the co-ordinates of the point P(t, y), and hence the equation for the linear bound.

There are:

$$\lfloor J_j / T_j \rfloor + 1 \tag{9}$$

invocations of τ_j released at time t=0, with subsequent releases occurring, for k=1,2…, at times:

$$\left(\lfloor J_j / T_j \rfloor + 1\right) T_j - J_j + (k-1) T_j \tag{10}$$

The number of subsequent releases that form part of the initial period of consecutive execution is therefore given by the largest value of k, such that the period of consecutive execution is equal to or exceeds the next release time of the task:

$$\left(\lfloor J_j / T_j \rfloor + 1\right) C_j + (k-1) C_j \geq \left(\lfloor J_j / T_j \rfloor + 1\right) T_j - J_j + (k-1) T_j \tag{11}$$

Simplifying, we have:

$$k \leq J_j / (T_j - C_j) - \lfloor J_j / T_j \rfloor \tag{12}$$

and hence the largest such value of k is given by:

$$k = \lfloor J_j / (T_j - C_j) \rfloor - \lfloor J_j / T_j \rfloor \tag{13}$$

Combining Equations (9) and (13), the total number h, of invocations of τ_j that execute consecutively, is:

$$h = \lfloor J_j / (T_j - C_j) \rfloor + 1 \tag{14}$$

The y co-ordinate of the point P(t, y) is therefore $y = hC_j + C_j$, which expands to:

$$y = \left(\lfloor J_j / (T_j - C_j) \rfloor + 1\right) C_j + C_j \tag{15}$$

The t co-ordinate of the point P(t, y) is found by adding $T_j + C_j$ to the release time of final invocation that forms part of the initial consecutive execution (itself found by substituting the value of k from Equation (13) into Equation (10)).

$$t = \left(\lfloor J_j / (T_j - C_j) \rfloor + 1\right) T_j - J_j + C_j \tag{16}$$

The slope of the linear bound is U_j and so from Equations (15) and (16), the equation for the linear upper bound $I_j^{UB}(t)$ is:

$$I_j^{UB}(t) = U_j t + U_j J_j + C_j (1 - U_j) \tag{17}$$

Summing over all tasks with higher priorities than i, an upper bound on the total interference due to higher priority tasks executed in the time interval [0, t) is given by:

$$\sum_{\forall j \in hp(i)} I_j^{UB}(t) = t \sum_{\forall j \in hp(i)} U_j + \sum_{\forall j \in hp(i)} (U_j J_j + C_j (1 - U_j)) \tag{18}$$

4.2. Occupied time upper bound

An upper bound on the longest time that the processor takes to execute higher priority tasks and computation C at priority i, is given by the intersection of the line y=t and the line

$$y = C + t \sum_{\forall j \in hp(i)} U_j + \sum_{\forall j \in hp(i)} (U_j J_j + C_j (1 - U_j)) \tag{19}$$

The solution (intersection of these lines) is given by:

$$O_i^{UB}(C) = \frac{C + \sum_{\forall j \in hp(i)} (U_j J_j + C_j (1 - U_j))}{1 - \sum_{\forall j \in hp(i)} U_j} \tag{20}$$

Theorem 1: $O_i^{UB}(C)$ is an upper bound on the *worst-case occupied period* $O_i(C)$, where $O_i(C)$ includes computation C at priority i.

Proof: First, we note the following from Figure 1. At any time t, the interference upper bound $I_j^{UB}(t)$, given by Equation (17) is either:

(i) strictly greater than the actual interference due to task τ_j executed in the time interval [0, t),

or

(ii) equal to the actual interference due to task τ_j executed in the time interval [0, t), *and* task τ_j has just completed execution at the earliest possible time, and is not released again until some non-zero time interval has elapsed.

To show that $O_i^{UB}(C)$ is an upper bound on $O_i(C)$, there are two cases to consider:

1. $t = O_i^{UB}(C)$ is a time at which, for one or more higher priority tasks τ_j, $I_j^{UB}(t)$ is strictly greater than the actual computation executed by the processor on behalf of task τ_j in the interval [0, t). In this case, the total computation executed by the processor in the interval [0, t) is strictly less than the length of the interval and hence the processor is able to start or resume execution at priority i some time before $t = O_i^{UB}(C)$.
2. $t = O_i^{UB}(C)$ is a time at which, for *all* higher priority tasks τ_j, $I_j^{UB}(t)$ is equal to the actual computation executed by the processor on behalf of task τ_j in the interval [0, t), and *all* the higher priority tasks have just completed execution simultaneously at their earliest possible times. (Note, this is only actually possible when there is one higher priority task). Hence, there is an interval of non-zero length before any further release of a higher priority task, and so the processor is available to start or continue executing at priority i at time $t = O_i^{UB}(C)$.

In both cases, at or before time t=$O_i^{UB}(C)$ the processor is able to continue executing at priority i □

4.3. Pre-emptive scheduling

In this section, we use the worst-case occupied time upper bound derived in Section 4.2 to obtain an upper bound response time for tasks with arbitrary deadlines and release jitter that are scheduled pre-emptively.

Comparing the analysis for pre-emptive scheduling in Section 3.2, with that for co-operative scheduling in Section 3.3, specifically Equations (3) and (5), we note

that as $\lfloor x/y \rfloor + 1 \geq \lceil x/y \rceil$ analysis for co-operative scheduling with $F_i = 0$, can be used to provide pessimistic worst-case response times for purely pre-emptive scheduling. Hence the worst-case occupied time upper bound $O_i^{UB}(B_i + (q+1)C_i)$ for execution of computation $B_i + (q+1)C_i$ at priority i forms an upper bound on the length of the priority level-i busy period given by Equation (3). From Equation (20),

$$W_i^{UB}(q) = \frac{B_i + (q+1)C_i + \sum_{\forall j \in hp(i)} (U_j J_j + C_j(1-U_j))}{1 - \sum_{\forall j \in hp(i)} U_j} \tag{21}$$

is therefore an upper bound on the length of the busy period $w_i(q)$ for invocation q of task τ_i.

Hence, a response time upper bound $R_i^{UB}(q)$ for the qth invocation is given by:

$$R_i^{UB}(q) = W_i^{UB}(q) - qT_i \tag{22}$$

Combining Equations (4) and (22), an overall response time upper bound R_i^{UB} for task τ_i is given by:

$$R_i^{UB} = \max_{\forall q}(W_i^{UB}(q) - qT_i) \tag{23}$$

Comparing the response time upper bounds for invocations q and $q+1$, we have:

$$R_i^{UB}(q) - R_i^{UB}(q+1) = T_i - \frac{C_i}{1 - \sum_{\forall j \in hp(i)} U_j} \tag{24}$$

Now as the overall taskset utilisation is assumed to be less than or equal to one, the right hand side of Equation (24) is either zero or positive, and so $R_i^{UB}(q)$ is a monotonic non-increasing function of q ($R_i^{UB}(q) \geq R_i^{UB}(q+1)$). Hence:

$$\forall q \quad R_i^{UB}(0) \geq R_i^{UB}(q) \tag{25}$$

and so $R_i^{UB} = R_i^{UB}(0)$. A response time upper bound for task τ_i is therefore given by:

$$R_i^{UB} = \frac{B_i + C_i + \sum_{\forall j \in hp(i)} (U_j J_j + C_j(1-U_j))}{1 - \sum_{\forall j \in hp(i)} U_j} \tag{26}$$

We note that Equation (26) reverts to the response time upper bound given by Bini and Baruah in [3] in the case where both blocking and release jitter are zero.

4.4. Co-operative scheduling

In this section, we use the worst-case occupied time upper bound derived in Section 4.2 to obtain an upper bound response time for tasks with arbitrary deadlines and release jitter, scheduled co-operatively; that is pre-emptively with deferred pre-emption.

$O_i^{UB}(B_i + (q+1)C_i - F_i)$ is an upper bound on the worst-case occupied time given by Equation (5). Hence from Equation (20),

$$V_i^{UB}(q) = \frac{B_i + (q+1)C_i - F_i + \sum_{\forall j \in hp(i)} (U_j J_j + C_j(1-U_j))}{1 - \sum_{\forall j \in hp(i)} U_j} \tag{27}$$

is an upper bound on the length of the priority level-i occupied period $v_i(q)$, during which invocation q of task τ_i may be delayed from starting its final non-pre-emptable section.

A response time upper bound $R_i^{UB}(q)$ for the qth invocation is therefore given by:

$$R_i^{UB}(q) = V_i^{UB}(q) + F_i - qT_i \tag{28}$$

Combining Equations (8) and (28), a response time upper bound R_i^{UB} for task τ_i is therefore given by:

$$R_i^{UB} = \max_{\forall q}(V_i^{UB}(q) + F_i - qT_i) \tag{29}$$

Comparing the response times upper bounds for invocations q and $q+1$, again we have:

$$R_i^{UB}(q) - R_i^{UB}(q+1) = T_i - \frac{C_i}{1 - \sum_{\forall j \in hp(i)} U_j} \tag{30}$$

and so $R_i^{UB}(q) \geq R_i^{UB}(q+1)$. Hence:

$$\forall q \quad R_i^{UB}(0) \geq R_i^{UB}(q) \tag{31}$$

and therefore $R_i^{UB} = R_i^{UB}(0)$.

A response time upper bound for task τ_i, scheduled co-operatively, is therefore given by:

$$R_i^{UB} = \frac{B_i + C_i - F_i + \sum_{\forall j \in hp(i)} (U_j J_j + C_j(1-U_j))}{1 - \sum_{\forall j \in hp(i)} U_j} + F_i \tag{32}$$

4.5. Non-pre-emptive scheduling

Non-pre-emptive scheduling is a special case of co-operative scheduling, with the final non-pre-emptive section equal to the entire execution of the task ($F_i = C_i$). Hence:

$$R_i^{UB} = \frac{B_i + \sum_{\forall j \in hp(i)} (U_j J_j + C_j(1-U_j))}{1 - \sum_{\forall j \in hp(i)} U_j} + C_i \tag{33}$$

provides a response time upper bound for a non-pre-emptively scheduled task τ_i.

4.6. Sufficient schedulability test

The response time upper bounds for pre-emptive, and non-pre-emptive scheduling, are just special cases of the bound for pre-emptive scheduling with deferred pre-emption. Hence we can use Equation (32) to form a sufficient schedulability test for fixed priority systems complying with a very general system model. This system model permits tasks that are scheduled pre-emptively ($F_i = 0$), co-operatively ($0 < F_i < C_i$) effectively offering limited pre-emption points and

hence deferred pre-emption, or non-pre-emptively ($F_i = C_i$) on a task-by task basis. Further, tasks may have arbitrary deadlines, arbitrary release jitter, and access mutually exclusive shared resources.

In this system model, the blocking factor B_i is computed as the longest time that a lower priority task can spend executing either non-pre-emptively, or whilst holding a mutually exclusive resource shared with a task of priority *i* or higher. Note it is assumed that the pre-emption points offered by co-operatively scheduled tasks do not occur within critical sections where mutually exclusive shared resources are locked.

The sufficient test is given below:

$$\forall i \quad R_i^{UB} \le D_i - J_i$$

where

$$R_i^{UB} = \frac{B_i + C_i - F_i + \sum_{\forall j \in hp(i)} (U_j J_j + C_j (1 - U_j))}{1 - \sum_{\forall j \in hp(i)} U_j} + F_i \tag{34}$$

The sufficient schedulability test checks tasks in priority order, highest priority first, computing their response time upper bound, and then comparing it with their deadline less release jitter to determine schedulability.

By checking tasks in priority order, highest priority first, the summation terms can be computed incrementally, via addition to the summation term for the previous, higher priority task. In this way the set of response time upper bounds, and hence the sufficient schedulability test for all *n* tasks, can be computed in linear time; O(*n*) for all *n* tasks.

4.7. Example

We now give an example of the effectiveness of the response time upper bound, based on a simple taskset with parameters chosen to be representative of tasks from an automotive electronic control unit. Here task periods commonly range from 10ms to 1 second, with release jitter of typically 50-100ms when tasks are released due to the arrival of messages transmitted over the network. Our example taskset comprises six tasks, with an overall utilisation of 85.5%. The tasks have deadlines less than or equal to their periods, and non-zero values for blocking and release jitter. The task parameters are given in Table 2.

Table 2: Example taskset

	C_i	T_i	D_i	J_i	B_i	$D_i - J_i$	R_i	R_i^{UB}
τ_1	3	10	10	2	0	8	3	3
τ_2	15	100	50	5	10	45	37	40
τ_3	15	200	200	5	10	195	58	75
τ_4	40	400	400	50	20	350	153	191
τ_5	30	1000	500	50	50	450	282	404
τ_6	200	1000	1000	100	0	900	682	876

The final two columns of Table 2 compare the exact response times of each task with the response time upper bound given by Equation (34). In this example, calculation of the response time upper bounds is sufficient to show that the taskset is schedulable. In contrast, the adapted utilisation-based tests, described in Section 5.1, deem all except the highest priority task to be unschedulable.

5. Empirical investigation

In this section, we report on the results of an empirical investigation, examining the effectiveness of the response time upper bound in the context of pre-emptive scheduling.

In each experiment, we compared the response time upper bound with utilisation-based tests: the Liu and Layland bound [23], the Hyperbolic bound [4] and the RBound [15], and also with the standard exact test given by Equation (2). In each case, the schedulability tests were applied on a task-by-task basis. Hence the complexity of the tests was as follows: response time upper bound O(*n*), Liu and Layland bound O(*n*), Hyperbolic bound O(*n*), RBound O(*nlogn*), exact test pseudo-polynomial.

5.1. Adaptations to the utilisation-based tests

So that we could examine comparative performance for tasksets with blocking, release jitter, and deadlines less than or equal to periods, we adapted the utilisation-based tests to cater for these more complex system models.

The Liu and Layland bound states that task τ_i is schedulable if:

$$\sum_{j=1..i} \frac{C_j}{T_j} \le i(2^{1/i} - 1) \tag{35}$$

assuming that there is no blocking or release jitter, that deadlines are equal to task periods, and that the tasks are in rate monotonic priority order.

For tasksets where tasks have deadlines less than or equal to their periods, release jitter, and priorities in 'deadline minus jitter' monotonic priority order [27], we can apply the principles of sustainable schedulability analysis [29] to transform the taskset into one that complies with the Liu and Layland system model. Recognising that the minimum inter-release time for task τ_i is given by $D_i - J_i$, that $R_i \le D_i - J_i$ is required for schedulability, and that the tasks are in 'deadline minus jitter' monotonic priority order, task τ_i is schedulable if:

$$\sum_{j=1..i} \frac{C_j}{D_j - J_j} \le i(2^{1/i} - 1) \tag{36}$$

Further, separating out the term for task τ_i and including blocking as if it were additional execution of τ_i, we have:

$$\frac{C_i + B_i}{D_i - J_i} + \sum_{j=1..i-1} \frac{C_j}{D_j - J_j} \le i(2^{1/i} - 1) \qquad (37)$$

The adapted version of the Liu and Layland bound given by Equation (37) is referred to in our experiments as simply the 'Liu & Layland bound'. The Hyperbolic bound and RBound were similarly adapted to cater for less restrictive system models.

5.2. Task parameter generation

The task parameters used in our experiments were randomly generated as follows: Of the n tasks in each taskset, n/M tasks were assigned to each of M 'order of magnitude' ranges used (e.g. 1-10ms, 10-100ms, 100ms-1s, etc.). Task periods were then determined according to a uniform random distribution, from the assigned range. This was done to replicate the type of period distributions found in commercial real-time systems. For each utilisation level studied, the UUniFast algorithm [28] was used to determine individual task utilisation U_i, and hence task execution times, $C_i = U_i T_i$, given the previously selected task periods.

Unless otherwise stated, the experiments used the following default parameter settings. Each taskset comprised 24 tasks (n=24). Task deadlines were set equal to their periods ($D_i = T_i$), and release jitter and blocking times were set equal to zero ($B_i = 0$, $J_i = 0$). The default number of order of magnitude period ranges used was 2 (M=2), and the default taskset utilisation 60% (U = 60%). 10,000 tasksets were generated for each x-axis value plotted on the graphs.

5.3. Experiments

Experiment 1: This experiment investigated the efficiency of the response time upper bound with respect to taskset utilisation, and the spread of task periods. The taskset utilisation was varied from 50% to 97.5% in increments of 2.5%, and the number of period 'order of magnitude ranges' varied from 1 to 5. The other parameters took the default values: n=24, $D_i = T_i$, $B_i = 0$, $J_i = 0$.

In this experiment, we used the taskset *acceptance ratio* as a measure of sufficient schedulability test effectiveness. For a given schedulability test, the acceptance ratio is defined as the number of tasksets deemed schedulable by the test, divided by the number of tasksets that are actually schedulable according to an exact test. Hence any exact schedulability test has an acceptance ratio of 100%.

The dashed line in Figure 2 represents the Liu and Layland bound of 70.3% utilisation for 24 tasks. The other utilisation-based tests gave very similar results. The five solid lines in Figure 2 are for the response time upper bound, showing how its performance varies with the number M, of order of magnitude ranges used in task period generation. For the wide spread of task periods ($M \ge 2$), and large number of tasks ($n \ge 10$) typical of commercial real-time systems, the performance of the response time upper bound is significantly better than that of the utilisation-based tests. However, when the range of task periods is restricted to be within just one order of magnitude, the performance of the response time upper bound is generally inferior to that of the utilisation based tests.

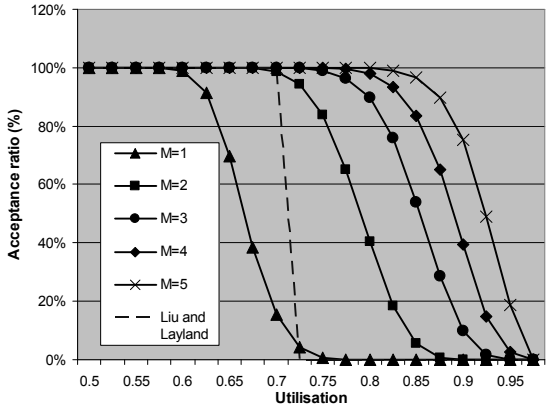

Figure 2

Experiment 2: This experiment investigated the efficiency of the tests with respect to tasksets with deadlines less than their periods. The deadlines of all the tasks were varied in lock step from 0.05 to 0.95 times their periods. The other parameters took the default values: n=24, M=2, $B_i = 0$, $J_i = 0$, U = 60%.

In this experiment (and all the subsequent ones), we used the *percentage of tasksets schedulable* as a measure of schedulability test effectiveness. This measure was computed by dividing the number of tasksets deemed schedulable by each test, by the total number of tasksets generated. Hence, in this case, it is necessary to compare the performance of the sufficient tests with that of the exact test, also plotted on the graphs.

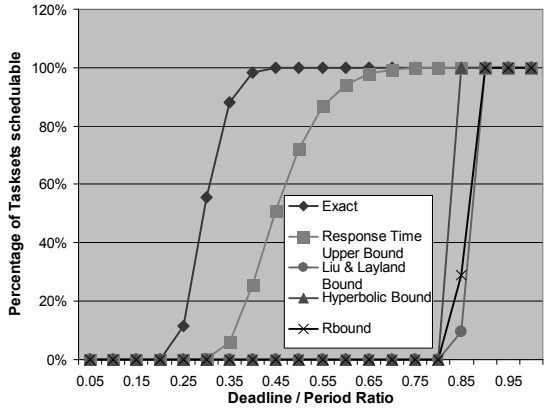

Figure 3

Figure 3 shows that using the exact test, the majority of the tasksets generated were schedulable with $D_i \ge 0.3T_i$ By comparison, using the response time upper bound, the majority of tasksets were

deemed schedulable with $D_i \geq 0.45T_i$. In contrast, the utilisation-based tests required $D_i \geq 0.8T_i$ before any of the tasksets were deemed schedulable.

This experiment shows that the response time upper bound is effective for applications where task deadlines are shorter than their periods.

Experiment 3: This experiment investigated the efficiency of the tests with respect to tasks with release jitter. The release jitter of all the tasks was varied in lock step from 0.05 to 0.95 times their periods. The other parameters took the default values: n=24, M=2, $D_i = T_i$, $B_i = 0$, $U = 60\%$.

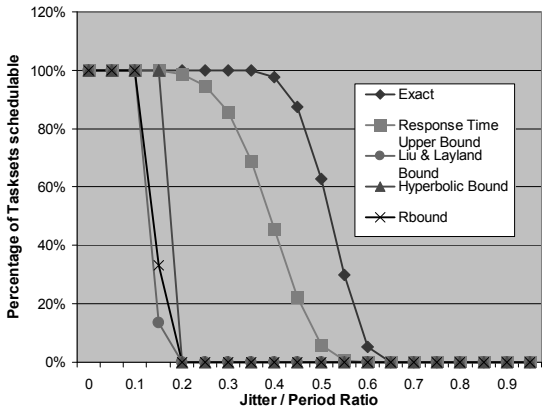

Figure 4

Figure 4, shows that using the exact test, the majority of the tasksets generated were schedulable with $J_i \leq 0.5T_i$. By comparison, using the response time upper bound, the majority of tasksets were deemed schedulable with $J_i \leq 0.35T_i$. In contrast, the utilisation-based tests required $J_i < 0.2T_i$ before any of the tasksets were deemed schedulable.

This experiment shows that the response time upper bound is effective for applications where tasks exhibit release jitter that greater than zero, but nevertheless relatively small in relation to their periods.

Experiment 4:

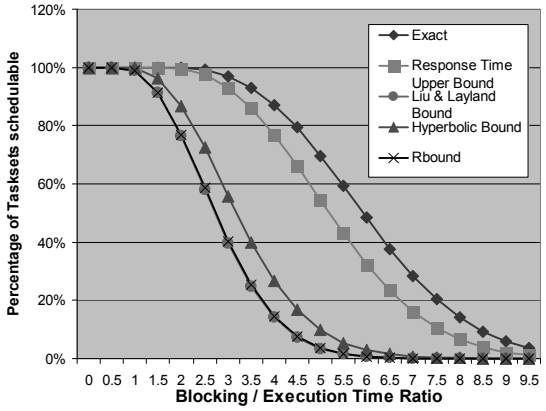

Figure 5

This experiment investigated the efficiency of the tests with respect to tasks with blocking. The blocking time of all the tasks was varied in lock step from 0.5 to 10 times their computation time, with the exception of the lowest priority task which had a blocking factor of zero. The other parameters took the default values: n=24, M=2, $D_i = T_i$, $B_i = 0$, $U = 60\%$.

Figure 5 shows that, for applications with a wide range of blocking factors, the performance of the response time upper bound is close to that of the exact test, and significantly better than that of the utilisation-based tests.

Experiment 5: This experiment investigated the efficiency of the tests with respect to tasksets with varying deadlines, release jitter and blocking, for a range of overall utilisation levels from 5% to 95%.

In this experiment, task deadlines, release jitter, and blocking factors were derived from task periods, deadlines, and execution times respectively, by applying random scaling factors chosen from a uniform distribution. Task deadlines were chosen in the range 0.5 to 1.0 times the task's period, release jitter in the range 0 to 0.5 times the task's deadline, and blocking factors in the range 0 to 1.0 times the task's execution time. The other parameters took the default values: n=24, M=2.

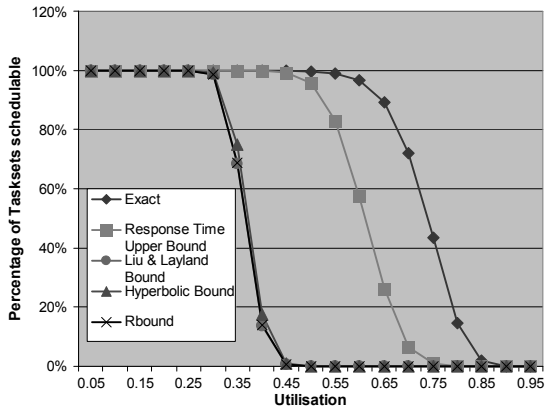

Figure 6

Figure 6 shows that for tasksets with varying parameters as described above, the majority of the tasksets generated were schedulable (according to the exact test) at 70% utilisation or below. By comparison, using the response time upper bound, the majority of tasksets were deemed schedulable at 60% utilisation or below. In contrast, using the utilisation-based tests, none of the tasksets with utilisation above 45% were deemed to be schedulable.

This experiment shows that the response time upper bound is effective for applications with a wide variety of task parameter settings.

Experiment 6: Considers the results of Experiment 5 in the context of using the response time upper bound to improve the efficiency of an exact test, by

determining, on a task-by-task basis, if an exact response time calculation is required.

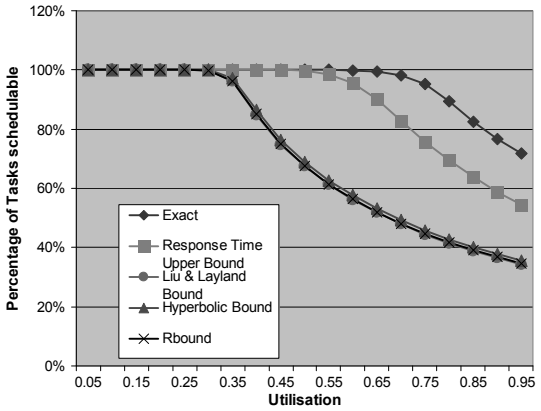

Figure 7

Figure 7 shows the percentage of *individual* tasks generated in Experiment 5 that were deemed schedulable by each of the tests. At 60% taskset utilisation, 95% of the tasks generated were deemed schedulable by the response time upper bound, and so only 5% of the tasks would require an exact response time calculation to determine their schedulability. At 75% taskset utilisation, more than three quarters of the tasks generated were deemed schedulable by the upper bound, and finally, even at the highest utilisation level of 95%, more than 50% of the tasks generated were schedulable according to the upper bound. These results indicate that a significant increase in exact schedulability test efficiency is possible by using the response time upper bound to determine on a task-by-task basis whether or not exact response time calculations are required.

In summary, Experiments 1-5 showed that for tasksets, resembling those present in commercial real-time systems, i.e. with a large number of tasks ($n \geq 10$), a spread of task periods spanning two or more orders of magnitude ($M \geq 2$), and tasks which exhibit blocking, release jitter and have deadlines less than or equal to their periods, the response time upper bound derived in this paper is significantly more effective than the utilisation-based tests, adapted from the Liu and Layland bound, the Hyperbolic bound or the RBound.

Further, the results of Experiment 6 indicate that using the response time upper bound to determine on a task-by-task basis whether exact response time calculations are required, is a highly effective means of improving exact schedulability test efficiency. Using the response time upper bound in this way means that time-consuming exact calculations are typically only required for a minority of tasks.

6. Summary and conclusions

In this paper we derived a closed form response time upper bound that is applicable to a very general system model, catering for tasks that are scheduled pre-emptively, co-operatively, and non-pre-emptively on a task-by-task basis. This system model places no restrictions on the relationship between task periods and their deadlines, or between task periods and their release jitter, allowing arbitrary values for these parameters. In addition, tasks are permitted to access mutually exclusive shared resources according to the Stack Resource Policy [2].

The response time upper bound forms a sufficient schedulability test which can be computed in linear time, O(n) for a set of n tasks, and is applicable to a wide range of fixed priority real-time systems, including both processors and networks [7].

The motivation for providing a generally applicable response time upper bound was two fold.

1. To improve the efficiency of exact schedulability tests.
2. To provide a simple and highly efficient, sufficient schedulability test that can be used stand-alone in a broad range of engineering contexts.

In interactive system design tools, system optimisation via search, and admission of new tasks into dynamic systems, schedulability test efficiency is a key consideration. Here, the response time upper bound can be used on a task-by-task basis to determine whether an exact response time computation is necessary in order to determine schedulability, significantly reducing the overall execution time of the schedulability test. This is illustrated in [8] for pre-emptively scheduled systems with task deadlines less than or equal to their periods, and no blocking or release jitter.

A number of scenarios can be envisaged where engineers may chose to use response time upper bounds in preference to a polynomial approximation or exact test. In the early stages of system design, when task or message parameters are only estimates, using response time upper bounds can be a pragmatic choice.

Although the bounds have a degree of pessimism, they have the advantage that they are continuous and differentiable in the task parameters. From an engineering perspective, this means that there can be no nasty surprises; increasing the jitter or execution time of a task by a small margin results in a commensurate small increase in its response time upper bound. In contrast, with exact analysis, the response time equations include either ceiling or floor functions, and so the exact response time is a discontinuous function of the task parameters. This means that a small increase in blocking, jitter or task execution times, or a small decrease in task periods can sometimes result in large increases in the exact response time.

Using exact response times, the sensitivity of a design to changes in task parameters can be difficult to estimate; whereas using response time upper bounds,

design sensitivity can be well understood by computing the first derivative of the bound with respect to each of the task parameters that may change.

Task admission to dynamic systems, is another case where using response time upper bounds may be preferable to exact analysis. Using a linear-time admission test here enables the operating system overheads to be tightly constrained.

Finally, when system optimisation (i.e. task and message allocation) is performed via search [9], the computational efficiency with which response time upper bounds can be calculated may be useful in narrowing the search to regions of interest, which can then, in a subsequent phase of the search, be explored in further detail using a slower but more precise exact response time calculation. Further, the fact that the response time upper bound is a continuous function of the task parameters may improve search efficiency. The effectiveness of this approach is something we aim to explore in future research.

In conclusion, the major contribution of this work is in providing for the first time, a closed form response time upper bound that is sufficiently general in its applicability, that it can be used in an engineering context to determine the schedulability of tasks and messages in complex real-world, real-time systems.

6.1. Acknowledgements and future work

This work was funded in part by the EU Frescor project. As part of this project, we aim to use the results of this research to improve the efficiency of the on-line admission tests and spare capacity allocation algorithms in the Frescor contract framework.

7. References

[1] N.C. Audsley, A. Burns, M. Richardson, and A.J.Wellings. "Applying new Scheduling Theory to Static Priority Pre-emptive Scheduling". *Software Engineering Journal*, 8(5) pp. 284-292, 1993.

[2] T.P. Baker. "Stack-based Scheduling of Real-Time Processes." *Real-Time Systems Journal* (3)1, pp. 67-100, 1991.

[3] E. Bini and S.K. Baruah. "Efficient Computation of Response Time Bounds under Fixed-priority Scheduling". *In Proceedings of the 15th conference on Real-Time and Network Systems*, pp. 95–104, Nancy, France, March 2007.

[4] E. Bini, G.C. Buttazzo, and G.M. Buttazzo. "Rate Monotonic Scheduling: The Hyperbolic Bound". *IEEE Transactions on Computers*, 52(7):933–942, July 2003.

[5] R.J. Bril, W.F.J. Verhaegh, and E-J.D. Pol. "Initial Values for On-line Response Time Calculations". *In Proceedings of the 15th Euromicro Conference on Real-Time Systems*, pp. 13–22, Porto, Portugal, July 2003.

[6] R.J. Bril, J.J. Lukkien, and W.F.J. Verhaegh. "Worst-Case Response Time Analysis of Real-Time Tasks under Fixed-Priority Scheduling with Deferred Preemption Revisited". *In Proceedings of the 19th Euromicro Conference on Real-Time Systems ECRTS*. pp. 269-279. July 2007.

[7] R.I. Davis, A. Burns, R.J. Bril, and J.J. Lukkien. "Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised". *Real-Time Systems*, Volume 35, Number 3, pp. 239-272. April 2007.

[8] R.I. Davis, A. Zabos, and A. Burns, "Efficient Exact Schedulability Tests for Fixed Priority Pre-emptive Systems" *IEEE Transactions on Computers* September 2008 (Vol. 57, No. 9) pp. 1261-1276.

[9] P. Emberson and I. Bate, "Minimising Task Migration and Priority Changes In Mode Transitions", *In proceedings 13th IEEE Real-Time And Embedded Technology And Applications Symposium* (RTAS 2007), pp. 158-167, 2007.

[10] M.S. Fineberg and O. Serlin, "Multiprogramming for hybrid computation", In *proceedings AFIPS Fall Joint Computing Conference*, pp. 1-13, 1967

[11] N. Fisher, C. H. Nguyen, J. Goossens, and P. Richard, "Parametric Polynomial-Time Algorithms for Computing Response-Time Bounds for Static-Priority Tasks with Release Jitters." *In Proceedings 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 377-385. Daegu Korea, August 2007.

[12] N. Fisher and S. Baruah. "A polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines." *In Proceeding of the 17th Euromicro Conference on Real-Time Systems*, pp. 117-126. Palma de Mallorca, Spain, July 2005.

[13] L. George, N. Rivierre, and M. Spuri. "Pre-emptive and non-pre-emptive real-time uni-processor scheduling. *Technical Report 2966*, Institut National de Recherche et Informatique et en Automatique (INRIA), France, September 1996

[14] M. Joseph and P.K. Pandya. "Finding Response Times in a Real-time System". *The Computer Journal*, 29(5):390–395, October 1986.

[15] S. Lauzac, R. Melhem, and D. Mosse. "An Improved Rate-monotonic Admission Control and its Applications". *IEEE Transactions on Computers*, 52(3):337–350, March 2003.

[16] J. Lehoczky. "Fixed priority scheduling of periodic task sets with arbitrary deadlines". In *Proceedings 11th IEEE Real-Time Systems Symposium*, pp. 201–209, IEEE Computer Society Press, December 1990.

[17] J.P. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behaviour". In *Proceedings of the 10th IEEE Real-Time Systems Symposium*, pp. 166–171, Santa Monica, CA, December 1989.

[18] J. Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic real-time tasks," *Performance Evaluation*, 2(4): 237-250, December 1982.

[19] C. L. Liu and J. W. Layland. "Scheduling algorithms for multiprogramming in a hard-real-time environment", *Journal of the ACM*, 20(1): 46-61, January 1973.

[20] S. Punnekkat, R. Davis, and A. Burns, "Sensitivity analysis of real-time task sets". In *Proceedings of the Asian Computing Science Conference*, pp. 72–82, Nepal, December 1997.

[21] P. Richard, J. Goossens, and N. Fisher, "Approximate Feasibility Analysis and Response-Time Bounds of Static-Priority Tasks with Release Jitters." *In Proceedings 15th International Conference on Real-Time and Network Systems*, pp. 105-112, March 2007.

[22] L. Sha, R. Rajkumar, and J.P. Lehoczky. "Priority inheritance protocols: An approach to real-time synchronization". *IEEE Transactions on Computers*, 39(9): 1175-1185, 1990.

[23] O. Serlin, "Scheduling of time critical processes". In *proceedings AFIPS Spring Computing Conference*, pp 925-932, 1972.

[24] M. Sjodin and H. Hansson. "Improved Response Time Analysis Calculations". *In Proceedings of the 19th IEEE Real-Time Systems Symposium*, pp. 399–408, Madrid, Spain, December 1998.

[25] K. W. Tindell. "Using Offset Information to Analyse Static Priority Pre-Emptively Scheduled Task Sets". *Technical Report YCS-92-182*. Dept. of Computer Science, University of York, UK, 1992.

[26] K.W. Tindell, , A. Burns, and A.J.Wellings. "An extendible approach for analyzing fixed priority hard real-time tasks". *Real-Time Systems.* Volume 6, Number 2, pp. 133-151 March 1994.

[27] A. Zuhily and A. Burns "Optimality of (D-J)-monotonic Priority Assignment". *Information Processing Letters*. Number 103, pp. 247-250, April 2007.

[28] E. Bini and G.C. Buttazzo. "Measuring the Performance of Schedulability tests". *Real-Time Systems*, 30(1–2):129–154, May 2005.

[29] S. Baruah and A. Burns, "Sustainable Scheduling Analysis". *In proceedings Real-Time Systems Symposium*, pp159-168, Dec. 2006.