# Exact Comparison of Fixed Priority and EDF Scheduling based on Speedup Factors for both Pre-emptive and Non-pre-emptive Paradigms

Robert I. Davis[1,2], Alan Burns[1], Sanjoy Baruah[3], Thomas Rothvoß[4], Laurent George[5], Oliver Gettings[1]
[1]Real-Time Systems Research Group, Department of Computer Science, University of York, York, UK.
[2]AOSTE Team, INRIA Paris-Rocquencourt, France.
[3]Department of Computer Science, University of North Carolina, Chapel Hill, NC 27599-317, Carolina, USA.
[4]Department of Mathematics, University of Washington, USA.
[5]Universite Paris-Est, LIGM, ESIEE Paris
rob.davis@york.ac.uk, alan.burns@york.ac.uk, baruah@cs.unc.edu, rothvoss@uw.edu, laurent.george@univ-mlv.fr

## Abstract

*This paper investigates the relative effectiveness of fixed priority (FP) scheduling in a uniprocessor system compared to Earliest Deadline First (EDF) scheduling. The quantitative metric used in this comparison is the processor speedup factor, defined as the factor by which processor speed needs to increase to ensure that any task set that is schedulable according to EDF can be scheduled using fixed priorities. In the pre-emptive case, exact speedup factors are known for sporadic task sets with implicit or constrained deadlines. In this paper, we derive exact speedup factors for both pre-emptive and non-pre-emptive fixed priority scheduling of arbitrary deadline sporadic task sets. We also show that the exact speedup factor for the pre-emptive case holds when tasks share resources according to the Stack Resource Policy / Deadline Floor Protocol.*

## Keyword

Fixed Priority; Earliest Deadline First; Scheduling; Sub-optimality; Resource Augmentation; Speedup factor; Pre-emptive; Non-pre-emptive; Uniprocessor.

## 1. Introduction

In this paper, we investigate the largest factor by which the processing speed of a uniprocessor needs to be increased, to ensure that any task set that was previously schedulable according to an optimal scheduling algorithm, such as EDF, is schedulable according to fixed priority scheduling. We refer to this resource augmentation factor as the *processor speedup factor* (Kalyanasundaram and Pruhs 1995).

### 1.1. Pre-emptive scheduling

Liu and Layland (1973) considered fixed priority pre-emptive (FP-P) scheduling of synchronous[1] task sets comprising independent periodic tasks, with bounded execution times, and deadlines equal to their periods. We refer to such task sets as *implicit-deadline* task sets. Liu and Layland (1973) showed that *rate monotonic* priority ordering (RMPO) is the optimal fixed priority assignment policy for implicit-deadline task sets, and that using RMPO, FP-P can schedule any implicit-deadline task set that has a total utilisation $U \leq \ln(2) \approx 0.693$. Liu and Layland (1973) also showed that Earliest Deadline First (EDF-P) is an optimal dynamic priority pre-emptive scheduling algorithm for implicit-deadline task sets, and that EDF-P can schedule any such task set that has a total utilisation $U \leq 1$.

Dertouzos (1974) showed that EDF-P is an optimal uniprocessor scheduling algorithm, in the sense that if a valid schedule exists for a task set, then the schedule produced by EDF-P will also meet all deadlines (Least Laxity First is another such optimal algorithm (Mok 1983)).

Research into real-time scheduling during the 1980s and early 1990s focused on lifting many of the restrictions of the Liu and Layland task model. Task arrivals were permitted to be sporadic, with known minimal inter-arrival times, (still referred to as periods), and task deadlines were permitted to be less than or equal to their periods (so called *constrained deadlines*) or less than, equal to, or greater than their periods (so called *arbitrary deadlines*).

Leung and Whitehead (1982) showed that *Deadline Monotonic*[2] Priority Ordering (DMPO) is the optimal fixed priority ordering for constrained-deadline task sets. Exact schedulability tests for FP-P scheduling of constrained-deadline task sets were introduced by Joseph and Pandya (1986), Lehoczky et al. (1988), and Audsley et al. (1993).

Lehoczky (1990) showed that DMPO is not optimal for task sets with arbitrary deadlines; however, an optimal priority ordering for such task sets can be determined in at most $n(n+1)/2$ task schedulability tests using the Optimal Priority Assignment (OPA) algorithm[3] given by Audsley (1991, 2001). Exact schedulability tests for task sets with arbitrary deadlines were developed by Lehoczky (1990), and Tindell et al. (1994).

[1] A task set is synchronous if all of its tasks share a common release time.

[2] *Deadline monotonic* priority ordering assigns priorities in order of task deadlines, such that the task with the shortest deadline is given the highest priority.

[3] This algorithm is optimal in the sense that it finds a schedulable priority ordering whenever such an ordering exists.

Exact schedulability tests for constrained and arbitrary-deadline task sets scheduled using Earliest Deadline First pre-emptive scheduling (EDF-P) were introduced by Baruah et al. (1990a, 1990b). Subsequently, exact tests for EDF-P have been developed by George and Hermant (2009) and Zhang and Burns (2009) that are more efficient in practice.

### 1.2. **Resource sharing**

In pre-emptive systems, concurrency control protocols are required to ensure that jobs access shared resources in mutual exclusion, otherwise data corruption, or erroneous behaviour of hardware devices could ensue. A number of concurrency control protocols have been developed, these include the Priority Inheritance Protocol (PIP) and the Priority Ceiling Protocol (PCP) (Sha et al. 1990) originally developed for FP-P scheduling and the Stack Resource Policy (SRP) (Baker, 1991), which is applicable to both FP-P and EDF-P scheduling. Baker (1991) initially provided a sufficient schedulability test for EDF-P and SRP. Later, Spuri (1996) modified the exact test for EDF-P scheduling to account for resource locking under SRP.

Exact tests for FP-P scheduling with either PCP or SRP where introduced by Audsley et al. (1993) for constrained deadline task sets, and Tindell et al. (1994) for arbitrary deadline task sets. Bletsas and Audsley (2006) showed that for FP-P scheduling with resource accesses according to PCP or SRP, DMPO is optimal for tasks with constrained deadlines, and the OPA algorithm is optimal for tasks with arbitrary deadlines. (Note, here, optimality is with respect to a task model where resource access times are known, but no information is available about the phasing of resource accesses within the tasks).

Baruah (2006) showed that EDF-P and SRP is optimal in the weak sense that it can schedule any task set with resource accesses for which a feasible work-conserving schedule exists. (Again, optimality is with respect to a task model where no information is available about the phasing of resource accesses within the tasks).

Burns et al. (2014) introduced the Deadline Floor inheritance Protocol (DFP), which is equivalent to SRP in terms of schedulability, but has a simpler implementation.

### 1.3. **Non-pre-emptive scheduling**

Kim and Naghibdadeh (1980) and Jeffay et al. (1991) gave exact schedulability tests for implicit-deadline task sets under Earliest Deadline First non-pre-emptive (EDF-NP) scheduling. These tests were extended by George et al. (1996) to the general case of sporadic task sets with arbitrary deadlines.

While EDF-P is an optimal uniprocessor scheduling algorithm, in the non-pre-emptive case no *work-conserving*[4] algorithm is optimal. This is because in general it is necessary to insert idle time to achieve a feasible schedule. The interested reader is referred to the work of George et al. (1996) for examples of this behaviour. Howell and Venkatrao (1995) showed that for non-concrete[5] periodic task sets, the problem of determining a feasible non-pre-emptive schedule is NP hard. Further they showed that for sporadic[6], task sets no optimal on-line inserted idle time algorithm can exist. In other words, clairvoyance is needed to determine a feasible non-pre-emptive schedule.

While no work-conserving algorithm is optimal in the strong sense that it can schedule any task set for which a feasible non-pre-emptive schedule exists; George et al. (1995) showed that EDF-NP is optimal in the weak sense that it can schedule any task set for which a feasible work-conserving, non-pre-emptive schedule exists. Hence we can regard EDF-NP as an optimal work-conserving, non-pre-emptive scheduling algorithm, for sporadic task sets.

For fixed priority non-pre-emptive (FP-NP) scheduling of arbitrary-deadline task sets, George et al. (1996) derived an exact schedulability test based on the approach of Tindell et al. (1994) for the pre-emptive case. George et al. (1996) showed that unlike in the pre-emptive case, DMPO is not optimal for constrained-deadline task sets scheduled by FP-NP. Further, they showed that the Optimal Priority Assignment algorithm given by Audsley (1991) is applicable, and can be used to determine an optimal priority ordering for task sets with arbitrary-deadlines scheduled using FP-NP.

Subsequent research by Bril et al. (2009) has refined exact analysis of FP-NP, correcting issues of both pessimism and optimism, and extending the schedulability tests to co-operative scheduling where each task is made up of a number of non-pre-emptive regions.

### 1.4. **Sub-optimality and speedup factors**

Combining the result of Dertouzos (1974) with the results of Liu and Layland (1973), shows that the processor speedup factor required to guarantee that FP-P scheduling can schedule any implicit-deadline task set schedulable by EDF-P is $1/\ln(2) \approx 1.44270$.

Davis et al. (2009b) derived the exact speedup factor for FP-P scheduling of constrained-deadline task sets; $1/\Omega \approx 1.76322$ (where $\Omega$ is the mathematical constant defined by the transcendental equation $\ln(1/\Omega) = \Omega$, hence, $\Omega \approx 0.567143$).

Davis et al. (2009a, 2010) also provided preliminary results giving upper and lower bounds on the speedup factors for arbitrary deadline task sets in the pre-emptive case (Davis et al. 2009a) and for implicit, constrained, and

[4] An algorithm is work-conserving if it never idles the processor when there is a job ready to execute.

[5] A periodic task set is referred to as *non-concrete* if the times at which each task is first released are unknown, also sometimes referred to as having *arbitrary phasing*.

[6] Sporadic task sets represent a generalisation of non-concrete periodic task sets.

arbitrary deadline task sets in the non-pre-emptive case (Davis et al. 2010)

This paper includes preliminary material from (Davis et al. 2009a, 2010) which it builds upon. The main contribution of the paper is the derivation of exact speedup factors for arbitrary deadline task sets in both the pre-emptive and non-pre-emptive cases.

### 1.5. **Related work on average case sub-optimality**

This paper examines the sub-optimality of fixed priority scheduling in the worst-case, other research has examined its behaviour in the average-case.

Lehoczky et al. (1989) introduced the *breakdown utilisation* metric: A task set is randomly generated, and then all task execution times are scaled until a deadline is just missed. The utilisation of the scaled task set gives the breakdown utilisation. Lehoczky et al. (1989) showed that the average breakdown utilisation, for implicit-deadline task sets of large cardinality under fixed priority pre-emptive scheduling is approximately 88%, corresponding to a penalty of approximately 12% of processing capacity with respect to an optimal algorithm such as EDF-P.

Bini and Buttazzo (2005) showed that breakdown utilisation suffers from a bias which tends to penalise fixed priority scheduling by favouring task sets where the utilisation of individual tasks is similar. Bini and Buttazzo introduced the *optimality degree* metric, defined as the number of task sets in a given domain that are schedulable according to some algorithm *A* divided by the number that are schedulable according to an optimal algorithm. Using this metric, they showed that the penalty for using fixed priority-pre-emptive scheduling for implicit-deadline task sets is typically significantly lower than that assumed by determining the average breakdown utilisation.

### 1.6. **Organization**

The remainder of the paper is organized as follows. Section 2 describes the system model, terminology and notation used. Section 3 recapitulates fixed priority and EDF schedulability analysis for both pre-emptive and non-pre-emptive cases.

In Section 4, we first derive an exact speedup factor for FP-P scheduling of arbitrary deadline sporadic task sets for the special case where Deadline Monotonic Priority Ordering is used instead of an optimal priority assignment policy. We then extend this result, proving that this exact speedup factor is valid in the general pre-emptive case, assuming arbitrary deadlines and optimal priority assignment. Finally, we show that this result continues to hold when tasks share resources according to the Stack Resource Policy (Baker 1991).

In Section 5, we first derive upper and lower bounds on the speedup factor for FP-NP scheduling of implicit, constrained, and arbitrary deadline sporadic task sets. We then prove what the exact speedup factor is in the general non-pre-emptive case, assuming arbitrary deadlines and optimal priority assignment.

Section 6 concludes with a summary and directions for future work.

## 2. System model, terminology, and notation

In this paper, we consider the scheduling of a set of sporadic tasks (or *task set*) on a uniprocessor system. Each task set comprises a static set of $n$ tasks $(\tau_1 ... \tau_n)$, where $n$ is a positive integer. We assume that the index $i$ of task $\tau_i$ also represents the task priority used in fixed priority scheduling, hence $\tau_1$ has the highest fixed-priority, and $\tau_n$ the lowest.

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

Each task $\tau_i$ is characterized by its bounded worst-case execution time $C_i$, minimum inter-arrival time or period $T_i$, and relative deadline $D_i$. Each task $\tau_i$ therefore gives rise to a potentially infinite sequence of invocations (or *jobs*), each of which has an execution time upper bounded by $C_i$, an arrival time at least $T_i$ after the arrival of its previous invocation, and an absolute deadline that is $D_i$ after its arrival.

In an *implicit-deadline* task set, all tasks have $D_i = T_i$. In a *constrained-deadline* task set, all tasks have $D_i \le T_i$, while in an *arbitrary-deadline* task set, task deadlines are independent of their periods. The set of arbitrary-deadline task sets is therefore a superset of the set of constrained-deadline task sets, which is itself a superset of the set of implicit deadline task sets.

The *worst-case response time* $R_i$ of a task $\tau_i$ is given by the longest possible time from release of the task until it completes execution. Thus task $\tau_i$ is schedulable if and only if $R_i \le D_i$, and a task set is schedulable if and only if $\forall i \quad R_i \le D_i$. The *utilisation* of a task $\tau_i$ is given by its execution time divided by its period ($U_i = C_i / T_i$). The total utilisation $U$, of a task set is the sum of the utilisations of all of its tasks.

We assume a discrete time model with granularity $\Delta$.

The following assumptions are made about the behaviour of the tasks:

- The arrival times of the tasks are independent and unknown a priori, hence the tasks may share a common release time.
- Each task is released (i.e. becomes ready to execute) as soon as it arrives.
- The tasks are independent and so cannot block each other from executing by accessing mutually exclusive shared resources, with the exception of the processor in the case of non-pre-emptive scheduling.
- The tasks do not voluntarily suspend themselves.

A task is said to be *ready* if it has outstanding

computation awaiting execution by the processor.

Under the EDF-P scheduling algorithm, at any given time the ready task invocation (job) with the earliest absolute deadline is selected for execution by the processor. In contrast, under FP-P scheduling, at any given time the highest priority ready task is selected for execution by the processor.

Similarly, under the EDF-NP scheduling algorithm, at any time when a job completes or the processor is idle, then the ready job with the earliest absolute deadline is selected for execution by the processor. Under FP-NP scheduling, at any time when a job completes or the processor is idle, then the highest priority ready task is selected for execution by the processor.

A task set is said to be *schedulable* with respect to some scheduling algorithm and some system, if all valid sequences of jobs that may be generated by the task set can be scheduled on the system by the scheduling algorithm without any missed deadlines.

A task set is said to be *feasible* with respect to a given system if there exists some scheduling algorithm that can schedule all possible sequences of jobs that may be generated by the task set on that system without missing any deadlines. A scheduling algorithm is said to be *optimal* with respect to a system and a tasking model (e.g. implicit, constrained, or arbitrary deadline sporadic tasks) if it can schedule all of the task sets that comply with the tasking model and are feasible on the system.

A schedulability test is termed *sufficient*, with respect to a scheduling algorithm and system, if all of the task sets that are deemed schedulable according to the test are in fact schedulable on the system under the scheduling algorithm. Similarly, a schedulability test is termed *necessary*, if all of the task sets that are deemed unschedulable according to the test are in fact unschedulable on the system under the scheduling algorithm. A schedulability test that is both sufficient and necessary is referred to as *exact*.

In fixed priority scheduling, a priority assignment policy $P$ is said to be *optimal* with respect to some class of task sets (e.g. arbitrary-deadline), and some class of fixed priority scheduling algorithm (e.g. pre-emptive) if all task sets in the class that are schedulable under the scheduling algorithm using some other priority ordering policy are also schedulable using the priority assignment determined by policy $P$.

The algorithm given by Audsley (1991, 2001) reproduced below, is an optimal priority assignment algorithm for arbitrary-deadline sporadic task sets in both the pre-emptive and non-pre-emptive case.

```
for each priority level k, lowest first {
    for each unassigned task τ {
        if(τ is schedulable at priority k with all
            other unassigned tasks assumed to have
            higher priorities) {
            assign τ to priority k
            break (continue outer loop)
        }
    }
    return unschedulable
}
return schedulable
```

**Algorithm 1: Optimal Priority Assignment (OPA) Algorithm**

## 2.1. Speedup factors and speedup optimal task sets

**Definition 1:** Let $S$ be some arbitrary task set, now assume that $\alpha^A(S)$ is the *critical scaling factor*, that is the maximum factor by which the execution times of all of the tasks in $S$ can be scaled, such that the task set is schedulable under algorithm $A$. Similarly, let $\alpha^{OPT}(S)$ be the maximum scaling factor under an optimal algorithm of the same class as $A$. The speedup factor $f^A(S)$ for the task set is given by:

$$f^A(S) = \alpha^{OPT}(S) / \alpha^A(S) \quad (1)$$

**Definition 2:** Let $f^{OPT}(\Psi)$ be the lowest processor speed such that task set $\Psi$ is schedulable according to an optimal scheduling algorithm of the same *class*[7]. Assume that $f^A(\Psi)$ is similarly the lowest processor speed that will schedule task set $\Psi$ using scheduling algorithm $A$. The *processor speedup factor* $f^A$ for algorithm $A$ is given by the maximum increase in processor speed required over an optimal algorithm of the same class for any task set $\Psi$.

$$f^A = \sup_{\Psi}\left(f^A(\Psi) / f^{OPT}(\Psi)\right) \quad (2)$$

where $\Psi$ ranges over all task sets.

For any scheduling algorithm $A$, we have $f^A \geq 1$, with smaller values indicative of a more effective algorithm, and $f^A = 1$ implying that $A$ is an optimal algorithm.

**Definition 3:** A task set is said to be *speedup-optimal* with respect to a scheduling algorithm $A$ if it requires the processor to be speeded up by the processor speedup factor in order to be schedulable under algorithm $A$. Hence for a speedup-optimal task set $\Psi$, $f^A(\Psi) / f^{OPT}(\Psi) = f^A$.

We note that in some cases, strictly speaking a speedup-optimal task set as defined here may not exist; rather, there is a family of task sets that get closer and closer to speedup optimality as some parameter used in their definition approaches infinity. Nevertheless, for brevity we abuse the terminology somewhat and refer to speedup-optimal task sets.

[7] By a *class* of algorithm we mean for example pre-emptive scheduling algorithms.

## 3. Schedulability analysis

We now recapitulate schedulability analysis for both pre-emptive and non-pre-emptive, fixed priority and EDF scheduling for sporadic task sets with arbitrary deadlines.

### 3.1. **Schedulability analysis for FP-P**

Analysis of fixed priority pre-emptive scheduling makes use of the concept of a *priority level-i busy period*. This term refers to a continuous period of time $[t_1, t_2)$ during which jobs of tasks of priority *i* or higher, that were released at the start of the busy period at $t_1$, or during the busy period but strictly before its end at $t_2$, are either executing or ready to execute.

For fixed priority pre-emptively scheduled systems, where task deadlines are arbitrary, execution of one job of a task may not necessarily be completed before the next job is released. Hence a number of jobs of task $\tau_i$ may be present within a priority level-*i* busy period, with earlier jobs delaying the execution of later ones.

Tindell (1994) showed that the worst-case scenario for task $\tau_i$ occurs following a *critical instant* where $\tau_i$ is released simultaneously with all higher priority tasks, and subsequent releases of task $\tau_i$ and higher priority tasks then occur after the minimum permitted time intervals. The length $L_i$ of this longest priority level-*i* busy period can be found via the following fixed point iteration:

$$L_i^{m+1} = \sum_{\forall j \in hep(i)} \left\lceil \frac{L_i^m}{T_j} \right\rceil C_j \tag{3}$$

Iteration starts with an initial value guaranteed to be no larger than the minimum solution, for example $L_i^0 = C_i$, and ends as soon as $L_i^{m+1} = L_i^m$ whereupon $L_i = L_i^m$. The number of jobs $Q_i^P$ of task $\tau_i$ in the busy period is given by:

$$Q_i^P = \left\lceil \frac{L_i}{T_i} \right\rceil \tag{4}$$

In general it is necessary to compute the response times of all jobs of a task $\tau_i$ within the longest priority level-*i* busy period in order to determine the task's worst-case response time. The *completion time* $W_{i,q}^P$ of the *q*th job (where $q = 0$ is the first job) of task $\tau_i$, with respect to the start of the busy period, is given by the following fixed point iteration:

$$w_{i,q}^{m+1} = (q+1)C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_{i,q}^m}{T_j} \right\rceil C_j \tag{5}$$

Iteration starts with an initial value $w_{i,q}^0$, typically $w_{i,q}^0 = (q+1)C_i$, and ends when either $w_{i,q}^{m+1} = w_{i,q}^m$ in which case , $W_{i,q}^P = w_{i,q}^{m+1}$ or when $w_{i,q}^{m+1} - qT_i > D_i$ in which case job *q*, and hence task $\tau_i$ is unschedulable.

To find the worst-case response time of task $\tau_i$, completion times $W_{i,q}^P$ need to be calculated for jobs $q = 0,1,2,3,...Q_i^P - 1$. The worst-case response time of task $\tau_i$ is then given by:

$$R_i^P = \max_{\forall q=0,1,2...Q_i-1} (W_{i,q}^P - qT_i) \tag{6}$$

Task $\tau_i$ is schedulable provided that $R_i^P \le D_i$.

Equations (5) and (6) give an exact schedulability test for the FP-P scheduling of arbitrary-deadline task sets with any fixed priority ordering.

A simpler sufficient rather than exact schedulability test can be derived by considering the maximum amount of task execution at priority *i* and higher released within an interval of length $D_i$ starting with simultaneous arrival of all of the tasks. If all of this execution can be completed by $D_i$, then this indicates that the length of the longest priority level-*i* busy period is at most $D_i$, and hence that all invocations of $\tau_i$ released in that busy period meet their deadlines, and so $\tau_i$ is schedulable. This sufficient schedulability test is given by:

$$\sum_{\forall j=hep(i)} \left\lceil \frac{D_i}{T_j} \right\rceil C_j \le D_i \tag{7}$$

### 3.2. **Schedulability analysis for FP-NP**

First, we introduce the concept of priority a Δ-critical instant which is fundamental to analysis of FP-NP scheduling.

A Δ-*critical instant* for a task $\tau_i$ is where task $\tau_i$ is released simultaneously with all higher priority tasks, and subsequent releases of task $\tau_i$ and the higher priority tasks occur after the minimum permitted time intervals. Further, the minimum possible amount of time[8] Δ prior to this simultaneous release, a lower priority task $\tau_k$ is released, and this task has the longest execution time of any such lower priority task.

George et al. (1996) and Bril et al. (2009) showed that for fixed priority non-pre-emptive scheduling, the longest response time of a task $\tau_i$ occurs for some job of that task within the priority level-*i* busy period starting at a Δ-*critical instant*. Lemma 3 in (Bril et al. 2009) states that the worst-case length of a priority level-*i* busy period $A_i$ is given by the minimum solution to the following fixed point iteration:

$$A_i^{m+1} = B_i + \sum_{\forall j \in hep(i)} \left\lceil \frac{A_i^m}{T_j} \right\rceil C_j \tag{8}$$

Iteration starts with an initial value $A_i^0$ guaranteed to be no larger than the minimum solution, for example $A_i^0 = C_i$, and ends when $A_i^{m+1} = A_i^m$ whereupon $A_i = A_i^m$. Note, $B_i$ is the longest time that task $\tau_i$ and any higher priority tasks can be blocked from executing by lower priority tasks, and is given by:

$$B_i = \begin{cases} \max\limits_{\forall k \in lp(i)} (C_k - \Delta) & i < n \\ 0 & i = n \end{cases} \tag{9}$$

The number of jobs $Q_i^{NP}$ of task $\tau_i$ in the busy period is given by:

[8] Recall, we assume discrete time, with granularity Δ.

$$Q_i^{NP} = \left\lceil \frac{A_i}{T_i} \right\rceil \tag{10}$$

The *start time* $W_{i,q}^{NP}$ of the $q$th job (where $q = 0$ is the first job) of task $\tau_i$ measured with respected to the start of the Δ-critical instant is given by the minimum solution to the following fixed point iteration:

$$w_{i,q}^{m+1} = B_i + qC_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_{i,q}^m + \Delta}{T_j} \right\rceil C_j \tag{11}$$

Iteration starts with an initial value $w_{i,q}^0$, typically $w_{i,q}^0 = B_i + qC_i$, and ends when either $w_{i,q}^{m+1} = w_{i,q}^m$ in which case $W_{i,q}^{NP} = w_{i,q}^{m+1}$, or when: $w_{i,q}^{m+1} + C_i - qT_i > D_i$ in which case job $q$, and hence task $\tau_i$ is unschedulable.

To find the worst-case response time, the start times $W_{i,q}^{NP}$ need to be calculated for jobs $q = 0,1,2,3,...Q_i^{NP} - 1$. The worst-case response time of task $\tau_i$ is then given by:

$$R_i^{NP} = \max_{\forall q=0,1,2...Q_i^{NP}-1} (W_{i,q}^{NP} + C_i - qT_i) \tag{12}$$

Task $\tau_i$ is schedulable provided that $R_i^{NP} \le D_i$.

From (8), we can form the following simple sufficient test for FP-NP scheduling of arbitrary-deadline task sets. Each task $\tau_i$ is schedulable provided that:

$$B_i + \sum_{\forall k \in hep(i)} \left\lceil \frac{D_i}{T_k} \right\rceil C_k \le D_i \tag{13}$$

If (13) holds, then this indicates that the solution to the fixed point iteration of (8) must be $\le D_i$. As the worst-case length of a priority level-$i$ busy period is then $\le D_i$, it follows that the worst-case response time $R_i$ of task $\tau_i$ must also be $\le D_i$, and hence the task must be schedulable.

### 3.3. **Schedulability analysis for EDF-P**

The schedulability of an arbitrary-deadline task set under EDF-P scheduling can be determined via the processor demand bound function $h(t)$ given below:

$$h(t) = \sum_{i=1}^{n} \max\left(0, \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) C_i \tag{14}$$

Baruah et al. (1990a, 1990b) showed that a task set is schedulable under EDF-P if and only if $U \le 1$ and a quantity referred to as the processor $LOAD$ is $\le 1$ where the processor $LOAD^P$ (pre-emptive case) is given by:

$$LOAD^P = \max_{t>0}\left(\frac{h(t)}{t}\right) \tag{15}$$

Further, they showed that if $LOAD^P \le 1$ for all values of $t$ in the interval $(0, L]$, where $L$ is defined as follows, then the task set is schedulable.

$$L = \max\left(D_1, D_2 ... D_n, \max_{\forall i}\left((T_i - D_i)\frac{U}{1-U}\right)\right) \tag{16}$$

Note the length $L$ of the interval that needs to be examined is also limited to the length of the longest priority level-$n$ busy period. The only values of $t$ that need to be checked in the interval $(0, L]$ are those where $LOAD^P$ can change, i.e. $\forall i \quad t = kT_i + D_i$ for integer values of $k$.

Zhang and Burns (2009) used the fact that $h(t)$ is monotonically non-decreasing with respect to $t$ as the basis for a highly efficient exact schedulability test for EDF-P, called QPA. The QPA algorithm is reproduced below.

*QPA schedulability test for EDF-P*: An arbitrary deadline sporadic task set is schedulable according to EDF-P if and only if $U \le 1$ and the result of the following iterative algorithm is $h(t) \le D_{\min}$, where $D_{\min}$ is the smallest deadline of any of the tasks, and $d_i$ is an absolute task deadline, i.e. $\forall i \quad d_i = kT_i + D_i$ for integer values of $k$.

```
1  t = max{d_i | d_i ≤ L}
2  while( h(t) ≤ t ∧ h(t) > D_min ) {
3      if( h(t) < t )
4          { t = h(t) }
5      else
6          { t = max{d_i | d_i < t} }
7  }
8  if( h(t) ≤ D_min ) ⇒ task set is schedulable
9  else ⇒ task set is unschedulable
```

**Algorithm 2: QPA algorithm for EDF-P**

### 3.4. **Schedulability analysis for EDF-NP**

George et al. (1996) extended the schedulability test embodied in (14) and (15) to the non-pre-emptive case (EDF-NP scheduling) via the addition of a blocking factor $B(t)$.

$$B(t) = \begin{cases} \max\limits_{\forall i: D_i > t} (C_i - \Delta) & t < \max\limits_{i=1...n}(D_i) \\ 0 & t \ge \max\limits_{i=1...n}(D_i) \end{cases} \tag{17}$$

George et al. (1996) showed that an arbitrary-deadline task set is schedulable under EDF-NP if and only if $U \le 1$ and the processor $LOAD$ is $\le 1$, where the processor $LOAD^{NP}$ (non-pre-emptive case) is defined by:

$$LOAD^{NP} = \max_{t>0}\left(\frac{h(t) + B(t)}{t}\right) \le 1 \tag{18}$$

Further, they showed that if $U \le 1$, then the only values of $t$ that need to be checked are those in the interval $(0, L]$ that correspond to times when $h(t) + B(t)$ can change, i.e. $\forall i \quad t = kT_i + D_i$ for integer values of $k$, where $L$ is the length of the longest priority level-$n$ busy period.

We observe that the value of $h(t) + B(t)$ is monotonically non-decreasing with respect to $t$. This follows from the fact that $B(t)$ can only decrease at times $t = D_1, D_2, ... D_n$, and at each of these times, the maximum possible decrease is upper bounded by $C_1, C_2, ... C_n$ respectively. At exactly those same times; however, $h(t)$ increases by $C_1, C_2, ... C_n$, thus the sum $h(t) + B(t)$ is monotonically non-decreasing with respect to $t$. This observation means that the same argument and logic used by Zhang and Burns (2009) to derive QPA can be applied in the non-pre-emptive case. Thus a QPA schedulability test for task set scheduled using EDF-NP can be obtained

simply by replacing each occurrence of $h(t)$ in Algorithm 2 with $h(t)+B(t)$:

```
1  t = max{d_i | d_i ≤ L}
2  while( h(t) + B(t) ≤ t ∧ h(t) + B(t) > D_min ) {
3      if( h(t) + B(t) < t )
4          { t = h(t) + B(t) }
5      else
6          { t = max{d_i | d_i < t} }
7  }
8  if( h(t) + B(t) ≤ D_min ) ⇒ task set is schedulable
9  else ⇒ task set is unschedulable
```

**Algorithm 3: QPA algorithm for EDF-NP**

The QPA tests for EDF-NP can also be derived as a special case of the test with shared resources as described by Zhang and Burns (2011).

## 4. Exact Speedup factor for FP-P

In this section, we derive the exact processor speedup factor required for fixed priority pre-emptive scheduling of arbitrary deadline task sets assuming optimal priority assignment. First, we derive the exact speedup factor for the (non-optimal) case where deadline monotonic priority ordering (DMPO) is used in conjunction with arbitrary-deadline task sets. The intuition behind these derivations is provided via an example.

### 4.1. Speedup Factor Example

The following simple example illustrating the concept of processor speedup factor defined in section 2.1.

Consider the arbitrary-deadline task set *S* comprising the two tasks defined in Table 1. The parameters of these tasks appear to have some unusual values; however, this is because they have been chosen so that the task set is just schedulable according to EDF-P, yet requires a speedup factor of 1.8 in order to be schedulable according to FP-P scheduling with priorities according to DMPO, which it should be noted is not optimal in this case.

**Table 1: Example Task set**

| Task | $C_i$ | $T_i$ | $D_i$ |
|---|---|---|---|
| $\tau_1$ | 1.8 | 2 | 16 |
| $\tau_2$ | 14.4 | $\infty$ | 17 |

We now show that task set *S* is schedulable according to EDF-P. Under EDF-P scheduling, the processor demand bound function $h(t)$ for task set *S* is the sum of the processor demand bound functions $h(t,\tau_1)$ and $h(t,\tau_2)$ for tasks $\tau_1$ and $\tau_2$ respectively, where $h(t,\tau_i)$ is the processor demand bound at time *t* for a single task $\tau_i$, given below:

$$h(t,\tau_i)=\max\left(0,\left\lfloor\frac{t-D_i}{T_i}\right\rfloor+1\right)C_i \qquad (19)$$

Thus:

$$h(t,\tau_1)=\left\{\begin{matrix}0 & t<16\\ \left\lfloor\frac{t-16}{2}\right\rfloor 1.8+1.8 & t\geq 16\end{matrix}\right\} \qquad (20)$$

as $\lfloor x/y\rfloor \leq x/y$, we have:

$$h(t,\tau_1)\leq\left\{\begin{matrix}0 & t<16\\ \frac{1.8(t-16)}{2}+1.8 & t\geq 16\end{matrix}\right\} \qquad (21)$$

Similarly, the processor demand bound function for task $\tau_2$ is:

$$h(t,\tau_2)=\left\{\begin{matrix}0 & t<17\\ 14.4 & t\geq 17\end{matrix}\right\} \qquad (22)$$

Recall that any arbitrary-deadline task set is schedulable according to EDF-P, provided that:

$$LOAD^P=\max_{\forall t}\left(\frac{h(t)}{t}\right)\leq 1 \qquad (23)$$

Now, given the following:

(i) The value of $h(t)$ at times $t=16$, $t=17$, and $t=18$ are 1.8, 16.2 and 18 respectively.

(ii) From (20) and (22) the value of $h(t)/t$ at time $t=18$ is 1. Further, from (21) and (22) an upper bound on $h(t)/t$ at time $t=18$ is also 1.

(iii) From (21), the rate of increase of the upper bound on $h(t)/t$ for $t\geq 18$ is 0.9.

It follows that the maximum value of $h(t)/t$ occurs at time $t=18$. The processor LOAD of task set *S* is therefore 1, indicating that the task set is just schedulable according to EDF-P.

We now consider the schedulability of task set *S* when scheduled according to FP-P scheduling, with priorities assigned according to DMPO, on a processor that has been speeded up by a factor of 1.8. The parameters of the task set on this faster processor are given in Table 2. We refer to this task set as *V*.

**Table 2: Example Task set**

| Task | $C_i$ | $T_i$ | $D_i$ |
|---|---|---|---|
| $\tau_1$ | 1 | 2 | 16 |
| $\tau_2$ | 8 | $\infty$ | 17 |

Figure 1 illustrates the execution of task set *V* under FP-P scheduling, assuming a synchronous arrival sequence.

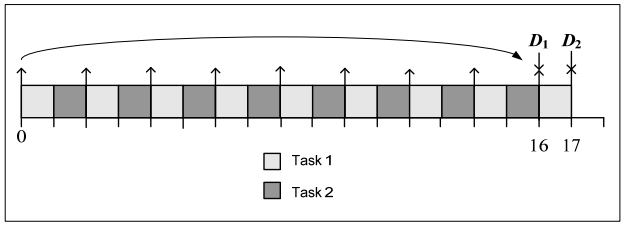


**Figure 1: FP-P Schedule**

We note that the worst-case response time of task $\tau_1$ is 1

and that of task $\tau_2$ is 16. Task set *V* is only just schedulable under fixed priority pre-emptive scheduling, using DMPO. Any reduction in processor speed would result in the task set being unschedulable. The processor speedup factor required is therefore 1.8.

### 4.2. Exact Speedup Factor for FP-P Scheduling with Deadline Monotonic Priority Ordering

We now derive the exact processor speedup factor required for the case where DMPO is used in conjunction with arbitrary-deadline task sets. Recall that DMPO is not optimal in this case (Lehoczky 1990); nevertheless, FP-P scheduling using DMPO is a simple combination of scheduling algorithm and priority assignment policy that is used in many real-time systems.

**Lemma 1:** An upper bound on the processor speedup factor for FP-P scheduling of arbitrary-deadline task sets using DMPO is 2.

**Proof:** Let *S* be any task set that is schedulable on a processor of unit speed according to an optimal scheduling policy such as EDF-P.

For each task $\tau_k$, in *S*, consider the processor demand bound during an interval of length $2D_k$. As task set *S* is schedulable according to EDF-P, it follows that:

$$\sum_{i=1}^{n} \max\left(0, \left\lfloor \frac{2D_k - D_i}{T_i} \right\rfloor + 1\right) C_i \le 2D_k \tag{24}$$

Next, consider task set *S* scheduled according to FP-P scheduling on a processor of twice the speed using DMPO. DMPO implies that $\forall i \le k \quad D_i \le D_k$.

From Equation (24) above, assuming a processor of twice the speed, and discarding the contribution from all tasks of lower priority than *k* we have:

$$\sum_{i=1}^{k} \max\left(0, \left\lfloor \frac{2D_k - D_i}{T_i} \right\rfloor + 1\right) C_i \le D_k \tag{25}$$

As $\lfloor x \rfloor + 1 \ge \lceil x \rceil$ and $\forall i \le k \quad D_i \le D_k$ then:

$$\sum_{i=1}^{k} \left\lceil \frac{D_k}{T_i} \right\rceil C_i \le D_k \tag{26}$$

Equation (26) is recognisable as the sufficient schedulability test for task $\tau_k$ in an arbitrary-deadline task set *S*, scheduled under fixed priority pre-emptive scheduling (see (7) in section 3.1). Repeating the above argument for each task $\tau_k$ in *S* proves that the task set is schedulable on a processor of speed 2 under fixed priority pre-emptive scheduling using DMPO □

**Theorem 1:** An exact bound on the processor speedup factor for FP-P scheduling of arbitrary-deadline task sets using DMPO is 2.

**Proof:** Consider task set *V* with the following parameters on a processor of speed $f$:

$\tau_1$: $C_1 = 1/2k$, $T_1 = 1/k$, $D_1 = 1$

$\tau_2$: $C_2 = 1/2$, $T_2 = \infty$, $D_2 = 1 + 1/2k$

where $k$ is a positive integer, and task $\tau_1$ has a higher priority than task $\tau_2$ i.e. DMPO. The execution of task set *V* under FP-P scheduling is illustrated in Figure 2. (Note the similarity to the task set used as an example in section 4.1).

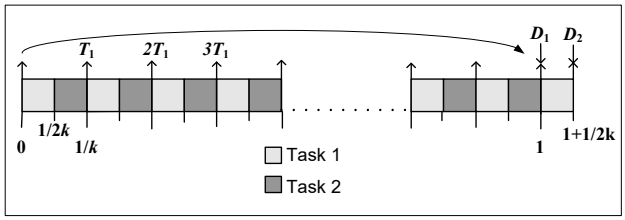


**Figure 2: FP-P Schedule**

We observe that with FP-P scheduling, any increase in the execution time of either task will cause task $\tau_2$ to miss its first deadline following simultaneous release of the two tasks.

We now consider the execution of task set *V* under EDF-P on a processor of unit speed. Let task set *S* be formed from task set *V* by increasing the execution times of tasks $\tau_1$ and $\tau_2$ by a scaling factor $f$ to form tasks $\tau_1'$ and $\tau_2'$, thus accounting for the reduction in processor speed.

We observe that $f = 2$ is an upper bound on the maximum scaling factor that could possibly result in a schedulable task set under EDF-P as this scaling factor results in task $\tau_1'$ having a utilisation of 100%.

Under EDF-P scheduling, the processor demand bound function $h(t)$ for task set *S* is the sum of the processor demand bound functions $h(t, \tau_1')$ and $h(t, \tau_2')$ for tasks $\tau_1'$ and $\tau_2'$ respectively.

$$h(t, \tau_1') = \begin{cases} 0 & t < 1 \\ \left\lfloor \dfrac{t - 1 + (1/k)}{(1/k)} \right\rfloor \dfrac{f}{2k} & t \ge 1 \end{cases} \tag{27}$$

as $\lfloor x/y \rfloor \le x/y$, we have the following upper bound:

$$h(t, \tau_1') \le \begin{cases} 0 & t < 1 \\ \dfrac{f(t-1)}{2} + \dfrac{f}{2k} & t \ge 1 \end{cases} \tag{28}$$

Similarly, the processor demand bound function for task $\tau_2'$ is:

$$h(t, \tau_2') = \begin{cases} 0 & t < 1 + (1/2k) \\ f/2 & t \ge 1 + (1/2k) \end{cases} \tag{29}$$

Recall that any arbitrary-deadline task set is schedulable according to EDF-P, provided that:

$$LOAD^P = \max_{\forall t}\left(\frac{h(t)}{t}\right) \le 1 \tag{30}$$

Now, given the following:

(i) The value of $h(t)/t$ at time $t = 1$ is $f/2k$.

(ii) An upper bound, from (28) and (29), on the value of $h(t)/t$ at time $t = 1 + (1/2k)$ is:

$$\frac{h(1 + (1/2k))}{(1 + (1/2k))} \le \frac{(f/2) + ((1 + (1/2k)) - 1)(f/2) + (f/2k)}{(1 + (1/2k))}$$

$$= \frac{f(k+(3/2))}{2(k+(1/2))} \quad (31)$$

(iii) The rate of increase of the upper bound on $h(t)/t$ for $t > 1+(1/2k)$ is $f/2$ (from (28)).

Then for values of $f \le 2$, the maximum value of the upper bound on $h(t)/t$ occurs at time $t = 1+1/2k$, therefore:

$$\max_{\forall t}\left(\frac{h(t)}{t}\right) = \frac{f(k+(3/2))}{2(k+(1/2))} \overset{\lim_{k\to\infty}}{=} \frac{f}{2} \quad (32)$$

From (32), the minimum value for the processor LOAD is achieved in the limit as $k \to \infty$, and this value is $f/2$. From Equation (32), for $k = \infty$, task set $V$ is schedulable according to EDF-P when its task execution times are scaled up by a factor of $f = 2$ to form task set $S$. Hence task set $S$ requires a processor speedup factor of 2 in order to be schedulable under FP-P scheduling with priorities in DMPO. As the processor speedup factor for FP-P scheduling of arbitrary-deadline task sets using DMPO is also upper bounded by 2 (Lemma 1), the exact processor speedup factor is 2 □

**Corollary 1:** Task set $S$ defined in the proof of Theorem 1 (with $k = \infty$), is a speedup-optimal task set for FP-P scheduling of arbitrary-deadline task sets using DMPO.

It is interesting to note that this speedup-optimal task set (requiring the largest speedup factor), includes a task $\tau_1$, with a deadline much larger than its infinitesimal period, and a task $\tau_2$, with a deadline much smaller than its infinite period.

### 4.3. Exact Speedup Factor for FP-P Scheduling using Optimal Priority Assignment

We now prove that the exact speedup factor for fixed priority pre-emptive scheduling of arbitrary deadline task sets, assuming optimal priority assignment is also 2. We do so by showing that this value is obtained for both upper and lower bounds on the speedup factor.

**Theorem 2:** An upper bound on the processor speedup factor for FP-P scheduling of arbitrary-deadline task sets using an optimal priority assignment algorithm is 2.

**Proof:** Follows directly from the fact that using an optimal priority assignment algorithm, FP-P scheduling can schedule any task set that is schedulable using DMPO. Hence the processor speedup factor required can be no greater with optimal priority assignment than the exact processor speedup factor given by Theorem 1 for DMPO □

**Theorem 3:** The exact processor speedup factor for FP-P scheduling of arbitrary-deadline task sets using an optimal priority assignment algorithm is 2.

**Proof:** Consider task set $S$ comprising $k+1$ tasks (where $k > 1$) with the following parameters on a processor of speed 1:

$\tau_1$ to $\tau_k$: $C_i = 1$, $D_i = 2k^2 - k$, $T_i = 2k$

$\tau_{k+1}$: $C_{k+1} = k^2$, $D_{k+1} = 2k^2$, $T_{k+1} = \infty$

(This task set is illustrated in Figure 3 for a relatively small value of $k$).

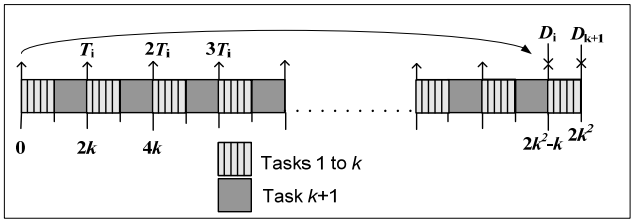


**Figure 3: FP-P Schedule**

Consider the feasibility of task set $S$ under FP-P. Applying Audsley's OPA algorithm, we must first find a task that is schedulable at the lowest priority level. If we place any one of the $k$ tasks labelled $\tau_1$ to $\tau_k$ at the lowest priority, then the response time of the first job of that task is given by:

$$R_i^{m+1} = 1 + k^2 + \left\lceil \frac{R_i^m}{2k} \right\rceil (k-1) \quad (33)$$

which converges to $2k^2 - k + 1$. This can be seen by first considering a schedule only including tasks $\tau_1$ to $\tau_{k-1}$. In such a schedule, in each interval of length $2k$ there is sufficient space for additional processing of $k+1$. By the end of $k-1$ such intervals (i.e. by time $2k^2 - 2k$), then the additional processing that could be accommodated is $k^2 - 1$. Hence to accommodate blocking of $k^2$ from task $\tau_{k+1}$, and execute the jobs of tasks $\tau_1$ to $\tau_{k-1}$ subsequently released at $2k^2 - 2k$ requires a further $k$ units of time. Finally, the first job of task $\tau_k$ completes at $2k^2 - k + 1$.

Alternatively, with task $\tau_{k+1}$ at the lowest priority, we have:

$$R_{k+1}^{m+1} = k^2 + \left\lceil \frac{R_{k+1}^m}{2k} \right\rceil k \quad (34)$$

which converges to $2k^2$, hence $\tau_{k+1}$ just meets its deadline. (Note we only need examine the response time of the first job of task $\tau_{k+1}$ since task $\tau_{k+1}$ has a constrained deadline). Thus the OPA algorithm is forced to place task $\tau_{k+1}$ at the lowest priority. It is easy to see that the remaining tasks (which are identical) are schedulable at the higher priority levels, since the length of the priority level-$k$ busy period is simply $k$ and ends before the next release (at $2k$) of any task.

Given that task $\tau_{k+1}$ completes at its deadline, it is clear that the processor cannot be slowed down by any factor and the task set remain schedulable using FP-P with optimal priority assignment. Note that the utilisation of the task set is 0.5.

Next, consider the schedulability of task set $S$ under EDF-P on a processor of speed:

$$s = \frac{k^2 + 3k/2}{2k^2} \quad (35)$$

Under EDF-P scheduling, the processor demand bound

function $h(t)$ for task set $S$ is the sum of the processor demand bound functions $h(t,\tau_1...\tau_k)$ and $h(t,\tau_{k+1})$ for tasks $\tau_1$ to $\tau_k$ and task $\tau_{k+1}$ respectively.

$$h(t,\tau_1...\tau_k)=\begin{cases}0 & t<2k^2-k\\ \left(\left\lfloor\frac{t-(2k^2-k)}{2k}\right\rfloor+1\right)\frac{k}{s} & t\geq 2k^2-k\end{cases} \quad (36)$$

as $\lfloor x/y\rfloor \leq x/y$, we have the following upper bound on $h(t,\tau_1...\tau_k)$:

$$h(t,\tau_1...\tau_k)\leq\begin{cases}0 & t<2k^2-k\\ \frac{k(t-(2k^2-k))}{s(2k)}+\frac{k}{s} & t\geq 2k^2-k\end{cases} \quad (37)$$

The processor demand bound function for task $\tau_{k+1}$ is:

$$h(t,\tau_{k+1})=\begin{cases}0 & t<2k^2\\ \frac{k^2}{s} & t\geq 2k^2\end{cases} \quad (38)$$

Recall that any arbitrary-deadline task set is schedulable according to EDF-P, provided that:

$$LOAD^P=\max_{t>0}\left(\frac{h(t)}{t}\right)\leq 1 \quad (39)$$

Given the following:

(i) The value of $h(t)/t$ at time $t=2k^2-k$ is:

$$\frac{k}{st}=\frac{k.2k^2}{(k^2+3k/2)(2k^2-k)}=\frac{2k}{2k^2+2k-3/2}<1 \quad (40)$$

(ii) An upper bound, from (37) and (38), on the value of $h(t)/t$ at time $t=2k^2$ is:

$$\frac{k^2+3k/2}{st}=1 \quad (41)$$

(iii) The rate of increase of the upper bound on $h(t)/t$ for $t>2k^2$ is as follow (from (37)):

$$\frac{1}{2s}=\frac{2k^2}{2(k^2+3k/2)}<1 \quad (42)$$

Then the maximum value of the upper bound on $h(t)/t$ occurs at time $t=2k^2$, and is equal to 1, hence task set $S$ is schedulable on a processor of speed $s$ under EDF-P.

In the limit, as $k\to\infty$ then $s\to 1/2$ and thus task set $S$ requires a processor speedup factor of 2 in order to be schedulable under FP-P with optimal priority assignment.

Since the upper bound on the speedup factor required by FP-P to schedule any task set that is schedulable under EDF-P is 2 (from Theorem 2) and we have shown that there exists a task set that requires that speedup factor, then the exact value of the speedup factor is 2. □

### 4.4. Exact Speedup Factor for FP-P Scheduling with Shared Resources

Previously, we considered the processor speedup factor required such that any feasible arbitrary-deadline task set, that was schedulable under an optimal algorithm such as EDF-P, could be guaranteed to be schedulable using FP-P scheduling. We now consider how changes to the task model, allowing tasks to access mutually exclusive shared resources according to the Stack Resource Policy (SRP) (Baker 1991), or to execute critical sections non-pre-emptively, affect the processor speedup factor required. (We note that the Deadline Floor inheritance Protocol (DFP) (Burns et al. 2014) for EDF has the same schedulability conditions as SRP, and hence our results also apply to comparisons between FP+SRP and EDF+DFP).

Recall that EDF-P + SRP is only optimal in the weak sense, that is with respect to work conserving scheduling algorithms (Baruah 2006), see section 1.2. We now determine the processor speedup factor required such that an arbitrary-deadline task set that is deemed schedulable by EDF-P + SRP according to the schedulability test given in (43) below, is guaranteed to be schedulable under FP-P + SRP scheduling using DMPO and hence also schedulable using optimal priority assignment.

We assume the same basic task model described in section 2; however, we relax the restriction on task independence. We assume that each task $\tau_k$ may access shared resources in mutual exclusion according to the Stack Resource Policy (Baker 1991). We define $B'(t)$ to be the maximum time for which any task $\tau_A$ with relative deadline $D_A>t$ may hold a resource needed by any other task $\tau_B$ with relative deadline $D_B\leq t$.

Under fixed priority scheduling, the *blocking factor* $B'_k$ for priority level *k*, is given by the maximum time for which a task of priority lower than *k* can access a resource that is shared with a task of priority *k* or higher.

Note in this section we use $B'(t)$ and $B'_k$ (as distinct from $B(t)$ and $B_k$) to distinguish blocking due to shared resources rather than non-pre-emptive scheduling.

A sufficient condition for task set schedulability under EDF + SRP was given by Spuri (1996):

$$\max_{t>0}\left(\frac{B'(t)+h(t)}{t}\right)\leq 1 \quad (43)$$

where $h(t)$ is the processor demand bound function given by (14). Spuri (1996) showed that the only values of $t$ that need to be checked are those corresponding to task deadlines within a synchronous busy period starting at time $t=0$.

With FP-P + SRP, the sufficient schedulability test for each task $\tau_k$ in an arbitrary-deadline task set, given by (7) can be extended to include the blocking factor as follows:

$$B'_k+\sum_{i\in hep(k)}\left\lceil\frac{D_k}{T_i}\right\rceil C_i\leq D_k \quad (44)$$

**Lemma 2:** An upper bound on the processor speedup factor required such that FP-P + SRP scheduling, using DMPO can schedule any arbitrary-deadline task set deemed schedulable under EDF-P + SRP by (43), is 2.

**Proof:** Follows a similar approach to the proof of Lemma 1. Let *S* be any task set that is schedulable according to (43) on

a processor of unit speed under EDF + SRP.

For each task $\tau_k$, in *S*, consider the processor demand bound and blocking factor for an interval of length $2D_k$. As task set *S* is schedulable according to EDF+SRP, it follows that:

$$\left( B'(2D_k) + \sum_{i=1}^{n} \max\left(0, \left\lfloor \frac{2D_k - D_i}{T_i} \right\rfloor + 1\right) C_i \right) \le 2D_k \qquad (45)$$

Next, consider task set *S* scheduled according to FP-P scheduling on a processor of twice the speed using DMPO. DMPO implies that $\forall i \le k \quad D_i \le D_k$.

From (45) above, assuming a processor of twice the speed, and separating out the contribution from all tasks of lower priority than *k* we have:

$$B'(2D_k) + \sum_{i \in lp(k)} \max\left(0, \left\lfloor \frac{2D_k - D_i}{T_i} \right\rfloor + 1\right) C_i + \sum_{i \in hep(k)} \max\left(0, \left\lfloor \frac{2D_k - D_i}{T_i} \right\rfloor + 1\right) C_i \le D_k \qquad (46)$$

As the tasks are in DMPO, we note that all of the tasks with priorities lower than *k* (i.e. in $lp(k)$) have deadlines $\ge D_k$. We now consider just the first and second terms in (46). Observe that the contribution to the second term from every task $\tau_i$ in $lp(k)$ with $D_i > 2D_k$ is zero. Further, the contribution from each task $\tau_i$ with $D_k \le D_i \le 2D_k$ is $C_i$. From the definition of $B'(t)$ it follows that the sum of the first two terms in (46) is $\ge B'_k$.

$$B'(2D_k) + \sum_{i \in lp(k) \wedge D_i \le 2D_k} C_i \ge B'_k \qquad (47)$$

Substituting $B'_k$ in place of the first two terms in (46), and noting for the third term that $\lfloor x \rfloor + 1 \ge \lceil x \rceil$ and $\forall i \in hep(k) \quad D_i \le D_k$ we obtain:

$$B'_k + \sum_{i \in hep(k)} \left\lceil \frac{D_k}{T_i} \right\rceil C_i \le D_k \qquad (48)$$

Equation (48) is identical to (44); the sufficient schedulability test for task $\tau_k$ in an arbitrary-deadline task set *S*, scheduled under FP-P + SRP. Repeating the above argument for each task $\tau_k$ in *S* proves that the task set is schedulable on a processor of speed 2 under FP-P + SRP scheduling using DMPO. □

**Theorem 4:** An exact bound on the processor speedup factor required such that FP-P + SRP scheduling, using DMPO can schedule any arbitrary-deadline task set deemed schedulable under EDF-P + SRP by (43), is 2.

**Proof:** Follows directly from Lemma 2 and the proof of Theorem 1 which shows that a speedup factor of 2 is necessary even without mutually exclusive resource access. □

**Theorem 5:** An exact bound on the processor speedup factor required such that FP-P + SRP scheduling, using optimal priority assignment can schedule any arbitrary-deadline task set deemed schedulable under EDF-P + SRP by (43), is 2.

**Proof:** Follows directly from Theorem 4, the fact that FP-P with optimal priority assignment dominates FP-P with DMPO, and the proof of Theorem 3 which shows that a speedup factor of 2 is necessary even without mutually exclusive resource access. □

## 5. Speedup factors for FP-NP

In this section, we derive upper and lower bounds on the processor speedup factor required for FP-NP scheduling using optimal priority assignment (Audsley 1991, 2001). These bounds are valid for sporadic and non-concrete[9] periodic task sets with implicit-, constrained-, and arbitrary-deadlines. Note that when we refer to the processor speedup factor for FP-NP scheduling, we mean as compared to EDF-NP, an optimal (in the weak sense (George et al. 1995)), work-conserving non-pre-emptive scheduling algorithm. We then derive an exact speedup factor valid for task sets with arbitrary deadlines.

First we present an example.

### 5.1. Speedup Factor Example

In the non-pre-emptive case, the concept of a speedup factor for a given task set *S* can be illustrated by means of the following example. Consider the task set *S* comprising the tasks defined in Table 3, with priorities assigned in the order that the tasks appear in the table (i.e. $\tau_A$ has the highest priority, and $\tau_D$ the lowest).

**Table 3: Example Task set**

| Task | $C_i$ | $D_i = T_i$ |
|---|---|---|
| $\tau_A$ | 1 | 6 |
| $\tau_B$ | 1 | 7 |
| $\tau_C$ | 1 | 8 |
| $\tau_D$ | $3 + \Delta$ | $\infty$ |

The worst-case arrival pattern for tasks $\tau_A$, $\tau_B$, and $\tau_C$ under FP-NP scheduling is shown in Figure 4. Note the 1st job of each task is shaded in grey, while the 2nd job of each task is un-shaded.

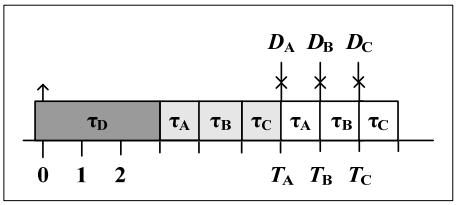


**Figure 4: FP-NP schedule**

Now consider the maximum factor by which the

[9] Recall that a periodic task set is referred to as non-concrete if the times at which each task is first released are unknown.

execution times of the tasks can be scaled and the task set remain schedulable according to FP-NP. This factor is $\alpha^{FP-NP}(S) = (6/5)^-$ (i.e. a value infinitesimally less than 6/5).

Figure 5 shows the FP-NP schedule for the scaled task set. Scaling by any larger factor, for example, a factor equal to 6/5 would result in the first job of task $\tau_C$ being unable to start executing before the 2nd job of task $\tau_A$ is released at time $t = 6$. It would then be further delayed by the 2nd job of task $\tau_B$, and hence fail to meet its deadline at time $t = 8$. In fact, there is no priority ordering which results in task set $S$, scaled by a factor of 6/5, being schedulable. This can be seen by considering the behaviour of the optimal priority assignment algorithm. While task $\tau_D$ is schedulable at the lowest priority, and can therefore be assigned priority 4, none of the other tasks are schedulable at priority 3.

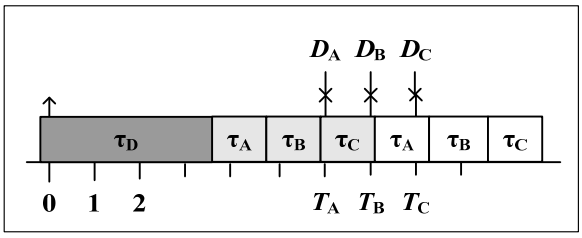


**Figure 5: FP-NP schedule, maximal scaling**

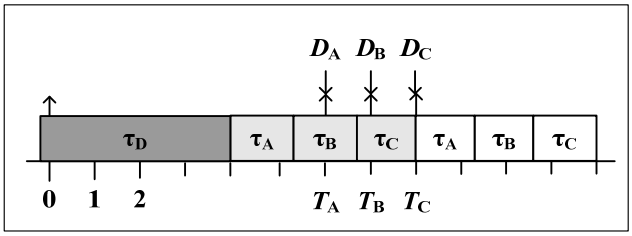


**Figure 6: EDF-NP schedule, maximal scaling**

With EDF-NP scheduling, the maximum scaling factor commensurate with task set $S$ remaining schedulable is $\alpha^{EDF-NP}(S) = 8/6$. Under EDF-NP, the first job of task $\tau_C$ has a later absolute deadline than the first jobs of tasks $\tau_A$ and $\tau_B$, and therefore executes after those jobs and after the first job of $\tau_D$ which is released at time $t = -\Delta$. The first job of task $\tau_C$ is not however delayed by the 2nd jobs of tasks $\tau_A$ and $\tau_B$, as these jobs have later absolute deadlines. With a scaling factor of 8/6, the first job of task $\tau_C$ just completes by its deadline (see Figure 6). Further analysis is required to prove that the scaled task set is schedulable under EDF-NP; however, as the priority level 3 busy period ends at $t = 12$, we need only check all deadlines in the interval [0, 12] to show schedulability. Note, Figure 6 shows the Δ-*critical instant* for tasks $\tau_A$, $\tau_B$, $\tau_C$, and all deadlines are met in the interval [0, 12]. Further, task $\tau_D$ is trivially schedulable as it has an infinite deadline, and the task set utilisation is less than 1.

Using (2), the speedup factor for the task set given in Table 3 is $f^{FP-NP}(S) = (8/6)/(6/5)^- = (40/36)^+ = (10/9)^+$ (i.e. a value infinitesimally larger than 10/9). In the next section, we generalise this example and show how task sets with a similar structure but with a large number of tasks require a much larger speedup factor.

## 5.2. Lower bound speedup factor for FP-NP

In this section, we derive a lower bound on the processor speedup factor required for FP-NP scheduling using optimal priority assignment (Audsley 1991, 2001). This lower bound is valid for sporadic and non-concrete periodic task sets with implicit-, constrained-, and arbitrary-deadlines. In the next section, we derive an upper bound with the same scope.

To derive a lower bound, we need only select a single task set and determine the required speedup factor for that task set. The task set $S$ that we use is a generalisation of the task set used as an example in the previous section. In this case, there are $n$ tasks, with the parameters given in Table 4. Tasks $\tau_1$ to $\tau_{n-1}$ are represented by $\tau_i$ in the first row of the table. All of these tasks have the same small execution time $\varepsilon << X$, and related periods/deadlines. Further, all of the tasks have periods equal to their deadlines, so this is an implicit-deadline task set. Task $\tau_n$ has an execution time of $X + \Delta$, where $X$ is a free variable that we can alter to maximise the required speedup factor.

**Table 4: Task parameters**

| Task | $C_i$ | $D_i = T_i$ |
|---|---|---|
| $\tau_i$ | $\varepsilon = \frac{1}{(n-1)}$ | $1 + X + \frac{(i-1)}{(n-1)}$ |
| $\tau_n$ | $X + \Delta$ | $\infty$ |

The execution of task set $S$ under FP-NP is depicted in Figure 7 below. Note, jobs of task $\tau_{n-1}$ are marked with an $\varepsilon$.

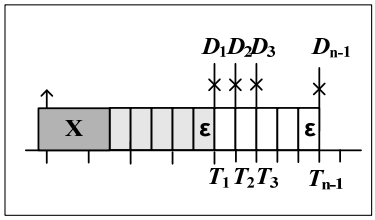


**Figure 7: FP-NP schedule**

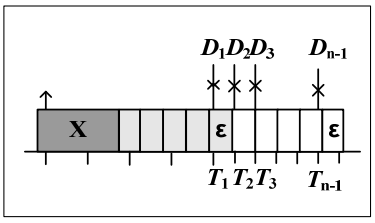


**Figure 8: FP-NP schedule, maximal scaling**

**Lemma 3:** The maximum factor $\alpha^{FP-NP}(S)$ by which the execution times of the tasks in task set *S* (Table 4) can be scaled and the task set remain schedulable according to FP-NP is given by:

$$\alpha^{FP-NP}(S) = \left(\frac{1+X}{1+X-\varepsilon}\right)^{-} \underset{\varepsilon\to 0}{=} 1^{+} \tag{49}$$

Figure 8 depicts the FP-NP schedule for the scaled task set.

**Proof:** Scaling by a factor equal to $(1+X)/(1+X-\varepsilon)$ would result in the first job of task $\tau_{n-1}$ being unable to start executing before the 2nd job of task $\tau_1$ is released at time $t = 1+X$ It would then be further delayed by the 2nd jobs of tasks $\tau_1$ to $\tau_{n-2}$, and hence fail to meet its deadline at time $t = 2+X-\varepsilon$. In fact, there is no priority ordering which results in task set *S*, scaled by a factor of $(1+X)/(1+X-\varepsilon)$, being schedulable. This can be seen by considering the behaviour of the optimal priority assignment algorithm (Algorithm 1), given the scaled task set: Task $\tau_n$ is schedulable at the lowest priority, and can therefore be assigned that priority. However, considering the remaining tasks in turn, none of them are schedulable at priority $n-1$. Task $\tau_1$ is not schedulable at priority $n-1$, as its 1st job would miss its deadline at $t = 1+X$. Task $\tau_2$ is not schedulable at priority $n-1$, as its 1st job is then unable to start before the 2nd job of task $\tau_1$ arrives, and so misses its deadline at $t = 1+X+\varepsilon$. In general, with a scaling factor of $(1+X)/(1+X-\varepsilon)$, for each task with index *i* from 2 to $n-1$, assuming that task $\tau_i$ is assigned priority $n-1$, ensures that the 1st job of task $\tau_i$ is unable to start before the 2nd job of task $\tau_{i-1}$ arrives, and so the 1st job of task $\tau_i$ misses its deadline.

By contrast, with a scaling factor of $(1+X)/(1+X-\varepsilon)^{-}$, task $\tau_{n-1}$ is schedulable at priority $n-1$, as it is able to start executing just prior to the arrival of the 2nd job of $\tau_1$ at $t = 1+X$. Further, with this scaling factor, all of the other tasks are schedulable with priorities assigned according to their indices (i.e. DMPO). This can be seen by checking the deadlines of all jobs up to the end of the priority level $n-1$ busy period, which occurs at:

$$t^A = (2+X)\left(\frac{(1+X)}{1+X-\varepsilon}\right)^{-} \tag{50}$$

As $t^A < 2(1+X) = 2D_1$, the priority level $n-1$ busy period comprises the 1st job of task $\tau_n$ and the 1st and 2nd jobs of tasks $\tau_1$ to $\tau_{n-1}$. All of these are schedulable (see Figure 8). The priority level-*n* busy period is of similar length, and hence task $\tau_n$ is trivially schedulable given its infinite deadline □

**Lemma 4:** The maximum factor $\alpha^{EDF-NP}(S)$ by which the execution times of the tasks in task set *S* (Table 4) can be scaled and the task set remain schedulable according to EDF-NP is given by:

$$\alpha^{EDF-NP}(S) = (1/\Omega)^{-} \tag{51}$$

**Proof:** There are two key conditions which limit the maximum scaling factor under EDF-FP (otherwise the task set would become unschedulable):

1. The 1st jobs of all tasks must be complete by the deadline of task $\tau_{n-1}$, $D_{n-1} = 2+X-\varepsilon$.
2. Utilisation of the scaled task set must not exceed 100%.

Considering the first condition, we have:

$$\alpha^{EDF-NP}(S) \le \frac{2+X-\varepsilon}{1+X} \tag{52}$$

The utilisation of the un-scaled task set is given by the sum of the utilisation of each task:

$$U = \sum_{i=1}^{n-1} \frac{1}{(n-1)} \frac{1}{(1+X+\frac{i-1}{n-1})} \tag{53}$$

The RHS of Equation (53) is recognisable as the left Riemann sum of the function 1/z, over the interval $[1+X, 2+X)$, hence:

$$U \underset{=}{^{n\to\infty}} \int_{1+X}^{2+X} \frac{1}{z} dz = \ln\left(\frac{2+X}{1+X}\right) \tag{54}$$

Thus, considering the second condition, we have:

$$\alpha^{EDF-NP}(S) \le 1/\ln\left(\frac{2+X}{1+X}\right) \tag{55}$$

As Equation (52) is monotonically non-increasing in *X* and tends to 2 for small *X*, and Equation (55) is monotonically non-decreasing in *X* and tends to 1/ln(2) for small *X*, then the maximum value is obtained when the RHSs of Equations (52) and (55) are equal, i.e. when:

$$\alpha^{EDF-NP}(S) = \frac{2+X-\varepsilon}{1+X} = 1/\ln\left(\frac{2+X}{1+X}\right) \tag{56}$$

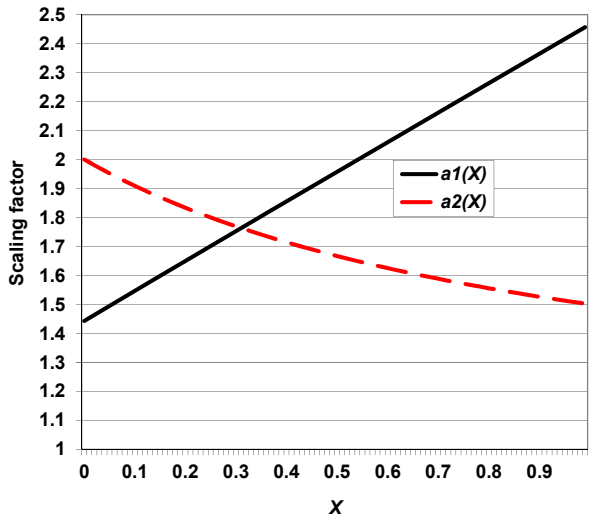


**Figure 9: Constraints on the scaling factor as a function of *X***

Figure 9 plots Equations (52) and (55) (labelled $a1(X)$ and $a2(X)$ respectively) against $X$. As $n \to \infty$, $\varepsilon \to 0$, the solution to Equation (56) is given by the intersection of the lines plotted in Figure 9, thus $\alpha^{EDF-NP}(S) = (1/\Omega)^{-} \approx 1.76322$, (where $\Omega$ is the mathematical constant defined by the transcendental equation $\ln(1/\Omega) = \Omega$, hence, $\Omega \approx 0.567143$). Further,

$$X = \frac{2\Omega - 1 - \varepsilon}{1 - \Omega} \approx 0.310232 \tag{57}$$

We now show that task set $S$ (Table 4) is schedulable under EDF-NP, when scaled by a factor of $\alpha^{EDF-NP}(S) = (1/\Omega)^{-}$. Proof is made significantly easier by the commonality between task set $S$ and the speedup-optimal task set $V$ for the constrained-deadline case of FP-P scheduling, described by Davis et al. (2009b) in Theorem 2 of that paper. In fact, tasks $\tau_1$ to $\tau_{n-1}$ are identical in these two task sets, only task $\tau_n$ differs. In task set $V$, the parameters of task $\tau_n$ are: $C_n = X$, $D_n = 2 + X$, and $T_n = \infty$, whereas in task set $S$, the parameters of task $\tau_n$ are: $C_n = X + \Delta$, and $D_n = T_n = \infty$ Theorem 4 given in the paper by Davis et al. (2009b) proves that task set $V$ is schedulable under EDF-P when scaled by a factor of $1/\Omega$. Hence for task set $V$,

$$LOAD^{P} = \max_{\forall t}\left(\frac{h(t)}{t}\right) \le 1 \tag{58}$$

We make use of this result to show that task set $S$, scaled by a factor of $(1/\Omega)^{-}$ is schedulable under EDF-NP. As tasks $\tau_1$ to $\tau_{n-1}$ are identical, their contribution to the processor demand bound $h(t)$ is the same for any time $t$. We now compare the contribution from task $\tau_n$ in each case. In the pre-emptive case, (task set V), $\tau_n$ contributes to $h(t)$ as follows:

$$h_n^P(t) = \begin{cases} 0 & 0 \le t < (2+X) \\ X/\Omega & t \ge (2+X) \end{cases} \tag{59}$$

whereas, in the non-pre-emptive case, (task set $S$), $\tau_n$ contributes only to the blocking factor:

$$B(t) = (X/\Omega)^{-} \quad t \ge 0 \tag{60}$$

Recall that in the non-pre-emptive case, a task set is schedulable provided that $U \le 1$ and:

$$LOAD^{NP} = \max_{\forall t}\left(\frac{h(t) + B(t)}{t}\right) \le 1 \tag{61}$$

Comparing (58) and (61), and the contributions of task $\tau_n$ in each case ((59) and (60)), it follows that (61) holds for all values of $t \ge (2+X)$ for task set $S$ scaled by a factor of $(1/\Omega)^{-}$. This is because, for all values of $t \ge (2+X)$ the value of $(h(t) + B(t))/t$ is the same as that for task set $V$, assuming both task sets are scaled by the same factor. To prove the schedulability of task set $S$ scaled by a factor of $(1/\Omega)^{-}$, it remains only to show that $(h(t) + B(t))/t \le 1$ for all values of $t$ in the interval $[0, (2+X))$. Here, we need only check values of $t$ that correspond to task deadlines. As $D_{n-1} < 2 + X < 2D_1$, this amounts to checking the 1st deadline of each of the $n-1$ highest priority tasks. At each of these deadlines $D_i$, we have:

$$h(D_i) = \left(\frac{1}{\Omega}\right)^{-} \sum_{k=1}^{n-1} \max\left(0, \left(\left\lfloor \frac{D_i - D_k}{T_k} \right\rfloor + 1\right)\right) \frac{1}{n-1} \tag{62}$$

as $\forall i,k \quad D_i = T_i$ and $D_i < 2D_k$ it follows that:

$$h(D_i) = \left(\frac{1}{\Omega}\right)^{-} \frac{i}{n-1} \tag{63}$$

Hence the scaled task set is schedulable provided that, $\forall i$ from 1 to $n-1$, $D_i \ge h(D_i) + B(D_i)$ that is:

$$1 + X + \frac{i-1}{n-1} \ge \left(\frac{1}{\Omega}\right)^{-} \left(X + \frac{i}{n-1}\right) \tag{64}$$

Substituting for $(1/\Omega)^{-} = (2 + X - \varepsilon)/(1 + X)$ and $\varepsilon = 1/(n-1)$, and rearranging, we have:

$$\left(1 + X + \frac{i-1}{n-1}\right)(1+X) \ge \left(2 + X - \frac{1}{n-1}\right)\left(X + \frac{i}{n-1}\right) \tag{65}$$

which simplifies to:

$$1 + \frac{i-1}{n-1} - \frac{2i}{n-1} + \frac{i}{(n-1)^2} \ge 0 \tag{66}$$

and then to:

$$\frac{n-i-2}{n-1} + \frac{i}{(n-1)^2} \ge 0 \tag{67}$$

For $n \ge 2$, the first term in inequality (67) is non-negative $\forall i$ from 1 to $n-2$, while the second term is always positive. Further, for $i = n-1$, the first and second terms cancel out, thus the inequality holds $\forall i$ from 1 to $n-1$. Task set $S$ is therefore schedulable according to EDF-NP when scaled by a factor of $(1/\Omega)^{-}$. □

**Theorem 6:** A lower bound on the speedup factor required

for FP-NP scheduling of an implicit-deadline task set is:

$$f^{FP-NP} = \frac{\alpha^{EDF-NP}(S)}{\alpha^{FP-NP}(S)} = \frac{(1/\Omega)^{-}}{1^{+}} = (1/\Omega)^{-} \qquad (68)$$

**Proof:** Follows from Lemmas 3 and 4 and the definition of the speedup factor (Definition 1). □

**Corollary 3:** We observe that as task set *S* is an implicit-deadline task set, and all implicit-deadline task sets are also constrained-deadline and arbitrary-deadline task sets, the lower bound of Theorem 6 applies to all three classes of task set.

## 5.3. Upper bound speedup factor for FP-NP

In this section, we derive an upper bound on the speedup factor required for FP-NP scheduling of arbitrary-deadline sporadic and non-concrete periodic task sets.

**Theorem 7:** An upper bound on the processor speedup factor required such that FP-NP scheduling, using optimal priority assignment can schedule any arbitrary-deadline sporadic or non-concrete periodic task set schedulable under EDF-NP according to (18), is 2.

**Proof:** Let *S* be any task set that is schedulable according to (18) on a processor of unit speed under EDF-NP. For each task $\tau_k$, in *S*, consider the processor demand bound and blocking factor for an interval of length $2D_k$. As task set *S* is schedulable according to EDF-NP, it follows that:

$$\left( B(2D_k) + \sum_{i=1}^{n} \max\left(0, \left\lfloor \frac{2D_k - D_i}{T_i} \right\rfloor + 1\right) C_i \right) \le 2D_k \qquad (69)$$

Next, consider task set *S* scheduled according to FP-NP scheduling on a processor of twice the speed using DMPO. DMPO implies that $\forall i \le k \quad D_i \le D_k$.

From (69) above, assuming speed 2, and separating out the contribution from all tasks of lower or equal priority to *k* we have:

$$B(2D_k) + \sum_{i \in lp(k)} \max\left(0, \left\lfloor \frac{2D_k - D_i}{T_i} \right\rfloor + 1\right) C_i + \sum_{i \in hep(k)} \max\left(0, \left\lfloor \frac{2D_k - D_i}{T_i} \right\rfloor + 1\right) C_i \le D_k \qquad (70)$$

As the tasks are in DMPO, we note that all of the tasks with lower priority than *k* (i.e. in $lep(k)$) have deadlines $\ge D_k$.

We now consider just the first and second terms in (70). Observe that the contribution to the second term from every task $\tau_i$ in $lep(k)$ with $D_i > 2D_k$ is zero. Further, there is a contribution from each task $\tau_i$ with $D_k \le D_i \le 2D_k$ of at least $C_i$. From the definition of $B(t)$ (17), the definition of $B_k$ (9), and the fact that the tasks are in DMPO, it follows that the sum of the first two terms in Equation (70) are $\ge B_k$, the blocking factor for FP-NP scheduling:

$$\max_{\forall i: D_i > 2D_k} \left\{ \begin{matrix} (C_i - \Delta) & if & 2D_k < \max_{\forall i}(D_i) \\ 0 & & otherwise \end{matrix} \right\} + \sum_{\forall i: D_k \le D_i \le 2D_k} C_i \ge B_k$$

$$\text{where } B_k = \begin{cases} \max\limits_{\forall i \in lp(k)} (C_i - \Delta) & k < n \\ 0 & k = n \end{cases} \qquad (71)$$

Substituting $B_k$ for the first two terms in (70) and transforming the third term by noting that $\lfloor x \rfloor + 1 \ge \lceil x \rceil$ and $\forall i \in hp(k) \quad D_i \le D_k$ we have:

$$B_k + \sum_{i \in hep(k)} \left\lceil \frac{D_k}{T_i} \right\rceil C_i \le D_k \qquad (72)$$

Equation (72) is identical to (13); the sufficient schedulability test for task $\tau_k$ in an arbitrary-deadline task set *S*, scheduled under FP-NP. Repeating the above argument for each task $\tau_k$ in *S* therefore proves that the task set is schedulable on a processor of speed 2 under FP-NP, with DMPO. As optimal priority assignment for FP-NP can schedule any task set that is schedulable using FP-NP with DMPO. □

**Corollary 4:** We observe that as the upper bound in Theorem 7 holds for arbitrary-deadline task sets, it must also hold for implicit-deadline, and constrained-deadline task sets.

## 5.4. Exact speedup factor for FP-NP for arbitrary deadline task sets

In this section, we derive the exact speedup factor required for FP-NP scheduling of arbitrary-deadline sporadic and non-concrete periodic task sets.

**Theorem 8:** The exact processor speedup factor required such that FP-NP scheduling, using optimal priority assignment can schedule any arbitrary-deadline sporadic or non-concrete periodic task set schedulable under EDF-NP according to (18), is 2.

**Proof:** We prove the theorem using a task set *V* derived from task set *S* used in the proof of Theorem 3. We split the final task of task set *S* into $k^2$ identical tasks each with an execution time of 1. Thus the task set *V* comprises $k^2 + k$ tasks with the following parameters on a processor of speed $s = 1$:

$\tau_1$ to $\tau_k$: $C_i = 1$, $D_i = 2k^2 - k$, $T_i = 2k$

$\tau_{k+1}$ to $\tau_{k+k^2}$: $C_j = 1$, $D_j = 2k^2$, $T_j = \infty$

Without loss of generality, we assume that the minimum processing time (e.g. a processor clock cycle) is given by $\Delta = 1$.

We now consider the schedulability of task set *V* under FP-NP scheduling assuming optimal priority assignment. Applying Audsley's OPA algorithm, we find that none of the tasks labelled $\tau_1$ to $\tau_k$ are schedulable at the lowest priority, since following synchronous release, the processor is occupied by higher priority tasks for an interval of time $t = 2k^2 - k$, hence the deadline of any such task would be reached before it was able to start executing.

By contrast, any of the $k^2$ tasks labelled $\tau_{k+1}$ to $\tau_{k+k^2}$ is schedulable at the lowest priority level since the longest

possible busy period (at the lowest priority level) is of length $2k^2$, which is equal to the deadline of those tasks. Since tasks $\tau_1$ to $\tau_k$ are schedulable at the highest *k* priority levels with $\tau_k$ having a worst-case response time of *k*, then the task set is schedulable with the tasks in the priority order given by their indices. (We note that other schedulable priority orders exist, but all require that one of the tasks with deadline $2k^2$ has the lowest priority).

Given that task $\tau_{k+k^2}$ completes at its deadline, it is clear that the processor cannot be slowed down by any factor and the task set remain schedulable using FP-NP with optimal priority assignment.

Next, consider the schedulability of task set *S* under EDF-NP on a processor of speed:

$$s = \frac{k^2 + 3k/2}{2k^2} \tag{73}$$

Recall that to prove schedulability under EDF-NP, we must show that:

$$LOAD^{NP} = \max_{t>0}\left(\frac{h(t)+B(t)}{t}\right) \le 1 \tag{74}$$

Since $\Delta = 1$ and all tasks have $C_i = 1$, then from (17) $\forall t \; B(t) = 0$. Further, since tasks $\tau_{k+1}$ to $\tau_{k+k^2}$ were obtained by decomposing the final task in task set *S* in the proof of Theorem 3 (pre-emptive case) into $k^2$ tasks with unit execution times with the same deadlines and periods, then it follows that $h(t)$ is identical to that for task set *S* in the pre-emptive case. Thus from the proof of Theorem 3 we have $h(t)/t \le 1$ and hence (74) holds and so task set *V* is also schedulable on a processor of speed *s* under EDF-NP. (Intuitively this is the case since the pre-emptive schedule for task set *S* and the non-pre-emptive schedule for task set *V* are equivalent).

In the limit, as $k \to \infty$ then $s \to 1/2$ and thus task set *V* requires a processor speedup factor of 2 in order to be schedulable under FP-NP with optimal priority assignment.

Since the upper bound on the speedup factor required by FP-NP to schedule any task set that is schedulable under EDF-NP is 2 (from Theorem 7) and we have shown that there exists a task set that requires this speedup factor, then the exact value of the speedup factor is 2 □

## 6. Conclusions and future work

We have examined the relative effectiveness of fixed priority and EDF scheduling, in both pre-emptive and non-pre-emptive cases. Our metric for measuring the relative effectiveness of fixed priority scheduling is a resource augmentation factor known as the processor speedup factor. In this case, the processor speedup factor is defined as the minimum amount by which the processor needs to be speeded up so that any task set that is schedulable by EDF-P (EDF-NP) scheduling is schedulable according to FP-P (FP-NP) scheduling using an optimal priority assignment policy.

Table 5 summarizes the new state-of-the-art in terms of analytical results regarding the speedup factors for FP-P and FP-NP scheduling. In the pre-emptive case, FP-P scheduling is compared to EDF-P which is an optimal pre-emptive uniprocessor scheduling algorithm. In the non-pre-emptive case, FP-NP is compared to EDF-NP which is an optimal (in the weak sense (George et al., 1995)), work-conserving non-pre-emptive scheduling algorithm.

**Table 5: Fixed priority scheduling speedup factors**

| Task set constraints | **Pre-emptive** Lower Bound | Upper Bound | **Non-pre-emptive** Lower Bound | Upper Bound |
|---|---|---|---|---|
| Implicit-deadline | $1/\ln(2) \approx$ 1.44269 | | $(1/\Omega)^- \approx$ **1.76322** | **2** |
| Constrained-deadline | $1/\Omega \approx$ 1.76322 | | $(1/\Omega)^- \approx$ **1.76322** | **2** |
| Arbitrary-deadline | **2** | | **2** | |

The major contributions of this paper (highlighted in bold in the above table) are in proving that:

(i) The exact processor speedup factor for fixed priority pre-emptive scheduling of sporadic or non-concrete periodic task sets with arbitrary deadlines and optimal priority assignment is 2.

(ii) The above result holds when tasks may access resources in mutual exclusion according to the Stack Resource Policy (Baker, 1991) (i.e. FP-P +SRP v. EDF-P +SRP) or the Deadline Floor inheritance Protocol (DFP) (Burns et al. 2014) (i.e. FP-P +SRP v. EDF-P +DFP).

(iii) The exact processor speedup factor for fixed priority non-pre-emptive scheduling of sporadic or non-concrete periodic task sets with arbitrary deadlines and optimal priority assignment is 2, when compared against non-pre-emptive EDF scheduling.

(iv) The upper and lower bounds on the processor speedup factor for fixed priority non-pre-emptive scheduling of sporadic or non-concrete periodic task sets with implicit or constrained deadlines are $(1/\Omega)^- \approx 1.76322$, and 2 respectively, when compared against non-pre-emptive EDF scheduling.

Thekkilakattil et al. (2013) quantified the speedup factor required for any task set schedulable under pre-emptive EDF to be schedulable under non-pre-emptive EDF. They derived an upper bound of 4 for the case where the largest execution time is not greater than the smallest deadline. This result can be combined with our result for FP-NP v. EDF-NP to give an upper bound of 8 on the speedup factor required for any task set schedulable under EDF-P to be schedulable under FP-NP, again assuming that the largest execution time is not greater than the smallest deadline. In future we aim to explore whether this bound can be tightened.

The seminal work of Liu and Layland (1973) characterises the maximum performance penalty incurred

when an implicit-deadline task set is scheduled using Rate-Monotonic, fixed priority pre-emptive scheduling instead of an optimal algorithm such as EDF-P. Davis et al. (2009b) gave an analogous result for constrained-deadline task sets. The research in this paper completes the exact quantification of the sub-optimality of fixed priority pre-emptive scheduling by determining the maximum performance penalty incurred when arbitrary deadline task sets are scheduled using that policy instead of an optimal pre-emptive scheduling algorithm such as EDF-P.

### 6.1. **Postscript**

As this article was going to press, further work on speedup factors for the case of FP-NP v. EDF-NP with implicit and constrained deadlines was published by von der Brüggen et al. (2015). They tightened the upper bounds in these cases from 2 to $1/\Omega \approx 1.76322$. Combined with the results given in this paper, and those previously published by Davis et al. (2009, 2010), this completes the set of six exact speedup factors for FP-P v. EDF-P and FP-NP v. EDF-NP for implicit, constrained, and arbitrary deadline task sets.


### Acknowledgements

This work was funded in part by the EPSRC projects TEMPO (EP/G055548/1) and MCC (EP/K011626/1). EPSRC Research Data Management: No new primary data was created during this study.


## References


Audsley N.C. (1991) "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times", Technical Report YCS 164, Dept. Computer Science, University of York, UK.

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

Audsley N.C. (2001) "On priority assignment in fixed priority scheduling", Information Processing Letters, 79(1): 39-44.

Baker T.P. (1991) "Stack-based Scheduling of Real-Time Processes." Real-Time Systems Journal (3)1, pages 67-100

Baruah S.K., Mok A.K., Rosier L.E. (1990a) "Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor". In Proceedings Real-Time Systems Symposium (RTSS), pages 182-190.

Baruah S.K., Rosier L.E., Howell R.R. (1990b) "Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic Real-Time Tasks on one Processor". Real-Time Systems, 2(4), pages 301-324.

Baruah, S.K, (2006) "Resource sharing in edf-scheduled systems: A closer look" In proceedings Real-Time Systems Symposium (RTSS), pages 379-387.

Bini E., Buttazzo G.C. (2005) "Measuring the Performance of Schedulability Tests", Real-Time Systems 30 (1-2), pages 129-154.

Bletsas, K., Audsley, N. (2006)."Optimal priority assignment in the presence of blocking" Information processing letters 99 (3), 83-86.

Bril, R.J., Lukkien, J.J., and Verhaegh, W.F. (2009) "Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred pre-emption". Real-Time Systems. 42, 1-3, pages 63-119.

Burns, A.; Gutierrez, M.; Aldea Rivas, M.; Gonzalez Harbour, M. (2014) "A Deadline-Floor Inheritance Protocol for EDF Scheduled Embedded Real-Time Systems with Resource Sharing," IEEE Transactions on Computers.

Davis R.I., Rothvoß T., Baruah S.K., Burns A. (2009b) "Exact Quantification of the Sub-optimality of Uniprocessor Fixed Priority Pre-emptive Scheduling." Real-Time Systems, Volume 43, Number 3, pages 211-258.

Davis, R.I., Rothvoß, T., Baruah, S.K., Burns, A. (2009a) "Quantifying the Sub-optimality of Uniprocessor Fixed Priority Pre-emptive Scheduling for Sporadic Task sets with Arbitrary Deadlines". In proceedings of Real-Time and Network Systems (RTNS), pages 23-31.

Davis R.I., George L., Courbin P. (2010) "Quantifying the Sub-optimality of Uniprocessor Fixed Priority Non-Pre-emptive Scheduling". In proceedings of Real-Time and Network Systems (RTNS), pages 1-10.

Dertouzos M.L. (1974) "Control Robotics: The Procedural Control of Physical Processes". In Proceedings of the IFIP congress, pages 807-813.

Fisher, N., Baker, T. P., Baruah, S. (2006) "Algorithms for Determining the Demand-Based Load of a Sporadic Task System". In Proceedings of Real-Time Computing Systems and Applications (RTCSA), pages 135-146.

George, L., Hermant, J. (2009) "A norm approach for the Partitioned EDF Scheduling of Sporadic Task Systems." In Proceeding Euromicro Conference on Real-Time Systems (ECRTS).

George, L., Rivierre, N., Spuri, M. (1996) "Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling", INRIA Research Report, No. 2966.

George, L., Muhlethaler, P., Rivierre, N. (1995) "Optimality and Non-Preemptive Real-Time Scheduling Revisited," Rapport de Recherche RR-2516, INRIA, Le Chesnay Cedex, France.

S-F. Hwang R-S He (2006) "Improving real-parameter genetic algorithm with simulated annealing for engineering problems". Adv. Eng. Softw. 37, 6 pages 406-418.

Howell, R.R., Venkatrao, M.K. (1995) "On non-preemptive scheduling of recurring tasks using inserted idle time", Information and computation Journal, Vol. 117, Number 1.

K. Jeffay, D. F. Stanat, C. U. Martel (1991) "On Non-Preemptive Scheduling of Periodic and Sporadic Tasks", In Proceedings Real-Time Systems Symposium (RTSS), pages 129-139.

Joseph M., Pandya P.K. (1986) "Finding Response Times in a Real-time System". The Computer Journal, 29(5), pages 390–395.

Kalyanasundaram B., Pruhs K. (1995) "Speed is as powerful as clairvoyance". In Proceedings of Symposium on Foundations of Computer Science, pages 214-221.

Kim, Naghibdadeh (1980) "Prevention of task overruns in real-time non-preemptive multiprogramming systems", In proceedings of Perf., Assoc. Comp. Mach. pages 267-276.

Leung J.Y.-T., Whitehead J. (1982) "On the complexity of fixed-priority scheduling of periodic real-time tasks". Performance Evaluation, 2(4), pages 237-250.

Lehoczky J. (1990) "Fixed priority scheduling of periodic task sets with arbitrary deadlines". In Proceedings Real-Time Systems Symposium (RTSS), pages 201–209.

Lehoczky J.P., Sha L., Ding Y. (1989) "The rate monotonic scheduling algorithm: Exact characterization and average case behaviour". In Proceedings Real-Time Systems Symposium (RTSS), pages 166–171.

Liu C.L., Layland J.W. (1973) "Scheduling algorithms for multiprogramming in a hard-real-time environment", Journal of the ACM, 20(1) pages 46-61.

Liu, J.W.S. (2000) “Real-Time Systems” Prentice Hall, ISBN-10: 0130996513.

Mok A.K. (1983) “Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment,” Ph.D. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts.

L. Sha, R. Rajkumar, J. Lehoczky, (1990) “Priority inheritance protocols: An approach to real-time synchronisation”. IEEE Transactions on Computers 39, 9, pages 1175–1185.

Spuri, M. (1996) “Analysis of Deadline Scheduled Real-Time Systems”. INRIA Technical Report No. 2772.

Thekkilakattil, A.; Dobrin, R.; Punnekkat, S. (2013) "Quantifying the Sub-optimality of Non-preemptive Real-Time Scheduling," In proceedings of Euromicro Conference on Real-Time Systems (ECRTS), pages 113,122.

Tindell K.W., Burns A., Wellings A.J. (1994) “An extendible approach for analyzing fixed priority hard real-time tasks”. Real-Time Systems. Volume 6, Number 2, pages 133-151.

von der Brüggen, G., Chen, J.J., Huang, W-H., (2015) “Schedulability and Optimization Analysis for Non-Preemptive Static Priority Scheduling Based on Task Utilization and Blocking Factors” In proceedings of Euromicro Conference on Real-Time Systems (ECRTS)

Zhang F., Burns A. (2009) "Schedulability Analysis for Real-Time Systems with EDF Scheduling," IEEE Transactions on Computers, pages 1250-1258.

Zhang F., Burns, A. (2011) “Schedulability Analysis of EDF Scheduled Embedded Real-Time Systems with Resource Sharing”. ACM Transactions on Embedded Computing Systems 9, 4, Article 39.