General Response Time Computation for the Deadline Driven Scheduling of Periodic Tasks*

Raymond Devillers and Joël Goossens
Département d’Informatique
Université Libre de Bruxelles
Brussels, Belgium
rdevil@ulb.ac.be, Joel.Goossens@ulb.ac.be

Abstract. In this paper we study the problem of scheduling hard real-time periodic task sets with a dynamic and preemptive scheduler. We will focus on the response time notion, its interest and its effective computation for the deadline driven scheduler. We present a general calculation procedure for determining the response time of the $k^{th}$ request of a task in asynchronous systems with general as well as arbitrary deadlines. Finally, we analyze the performance of the computation so defined, both in terms of time and memory requirements.

Keywords: hard real-time scheduling, periodic task set, synchronous systems, asynchronous systems, deadline driven scheduler, response time.

1. Introduction

In real-time systems, not only must the processes deliver a correct answer, but they must also do so in due time, generally expressed in the form of a deadline. This leads to interesting problems, and even in apparently simple cases, like systems composed of independent periodic tasks on a single preemptive processor, the behaviour of the schedules may be surprisingly complex. We shall thus consider a system of $n$ tasks $\tau_i$, each one being characterized by a tuple $(C_i, D_i, T_i, O_i)$, where $C_i$ is the (worst case) computation time requirement of each request of $\tau_i$, $D_i$ is the deadline delay of each of these requests, $T_i$ is their period and the offset $O_i$ is the arrival time of the first request (all these characteristics are usually mere natural integers). I.e.,

*The authors would like to thank the anonymous referees for their suggestions and encouragements.
the $k^{th}$ request of $\tau_i$ occurs at time $O_i + (k-1) \cdot T_i$, uses $C_i$ units of processor time and must be finished before or at time $O_i + (k-1) \cdot T_i + D_i$. The problem is to find good schedulers and to be able to state if all the deadlines will ever be met or not.

One generally distinguishes static schedulers, where each task receives a distinct priority beforehand, and dynamic schedulers, where each request of each task receives some priority, which may even change with time. Interesting sub-cases are synchronous systems, where $O_i = 0 \forall i$ (otherwise the system is said to be asynchronous, or offset free if the offsets are not fixed by the problem but may be chosen by the scheduler [4, 3]); late deadline systems, where $C_i \leq D_i = T_i \forall i$; general deadline systems, where $C_i \leq D_i \leq T_i \forall i$; and arbitrary deadline systems, where it is only required that $C_i \leq D_i \forall i$ (in this case many requests of a single task may be simultaneously active even if the system is feasible, i.e., all deadlines are met).

Synchronous late deadline systems with static schedulers are particularly popular in the literature and among practitioners, not because of their generality or efficiency, but simply because in this case, an easy-to-implement optimal scheduler is known (the rate monotonic scheduler, RMS for short, which gives higher priorities to lower periods [10, 14]) and there is a very simple and fast sufficient feasibility test based on the utilization factor $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$ (if $U < \ln 2$ the system is always schedulable, as well as if $U < n \cdot (\sqrt{2} - 1)$) [10]. If we want to be a bit more liberal and allow synchronous general deadline systems, we still know an optimal scheduler (the deadline monotonic scheduler, DMS for short, which gives higher priorities to lower deadline delays [9]) but there is no known simple and fast schedulability test; however, it can be shown that for each task the first request has the worst response time (this remains true for any static scheduling rule [10, 3], not only for DMS or RMS), and Audsley [1] and Tindell [15] have shown how to compute it reasonably fast without performing a true simulation, which leads to a necessary and sufficient feasibility test (also applicable for late deadline systems when $n \cdot (\sqrt{2} - 1) < U \leq 1$; if $U > 1$, the system is unfeasible on a monoprocessor anyway). If we now turn to synchronous arbitrary deadline systems, the situation is less favourable since DMS is not an optimal scheduling rule anymore; we do not even know an optimal static priority allocation (but checking all possibilities); and the first request of each task has not necessarily the worst response time [7], so that it may be necessary to check subsequent response times (by simulation or by direct computations [6]) till the first idle period or idle point [3, 5]. For asynchronous or offset free systems, the situation is similar: we do not know an optimal static priority/offset allocation (apart from checking all non-equivalent possibilities [3, 2] and computing many response times [3, 6]).

Better scheduling performances may be obtained with dynamic strategies (since static ones may be considered as a special case anyway), and a first curious feature is that one knows two optimal scheduling rules: the deadline driven scheduler (DDS for short) which attributes the CPU to the active request with the least deadline [10], and the least laxity first scheduler (LLF for short) which attributes the CPU to the active request with the least difference between the delay before the deadline and the CPU time still needed to complete the request [13, 12, 8]. Both are optimal in the sense that, whatever the offsets and the deadlines, if there is a scheduling rule
without deadline miss, the system is also feasible with them. While DDS and LLF are equivalent with respect to the feasibility of the systems, they generally behave quite differently; for instance DDS yields less preemptions than LLF (which may be important if, contrary to what is usually assumed in those theories, preemptions are not negligible for the performances of the system); this is due to the fact that DDS the relative priorities of two (active) requests do not change with time, while they do with LLF; that is why, in the following, we shall only consider the deadline driven scheduler. For late deadline systems there is a very simple feasibility check: a system is feasible if and only if $U \leq 1$. Unfortunately, even for synchronous general deadline systems, their feasibility with dynamic schedulers is generally intrinsically exponential in terms of the number of tasks. For instance, it is no longer true that for each task the first request has the worst response time, so that it is necessary to determine more response times to make sure that the system is feasible or not. Hence, we shall show here how to compute rather efficiently (in comparison with brute force simulation) any response time with DDS, whatever the offsets and deadlines.

We shall first consider the schedule of a very simple system in order to introduce our graphical conventions and some preliminary technical terms.

**Example 1.1.** Consider the following system composed of two tasks $\tau_1$ and $\tau_2$, with $\tau_1 = \{C_1 = 1, D_1 = 3, T_1 = 4, O_1 = 0\}$ and $\tau_2 = \{C_2 = 2, D_2 = 3, T_2 = 5, O_2 = 0\}$. Figure 1 corresponds to the schedule of this set using a deadline driven scheduler. In this figure (as in the next one), $\downarrow$ represents the arrival of a task request, $\bigcirc$ a deadline, and $\overline{a\rightarrow b}$ an execution of $c$ units between time slots $a$ and $b$ (included, numbered from 0) for some task; in the special case where $a = b$, we omit $b$ in our representation. From this schedule several remarks can be made.

- At time $t = 0$ two requests are active (i.e., they are waiting for the CPU in order to start/continue their execution) and their deadlines coincide (at time $t = 3$): the tie is broken here by giving a higher priority (and then the CPU) to the request of $\tau_1$.
- The priority of the various requests of the same task changes with time: for instance at time $t = 0$ the first request of $\tau_1$ has a higher priority than the first request of $\tau_2$, while at time $t = 16$, the fifth request of $\tau_1$ has a lower priority than the (end of the) fourth request of task $\tau_2$.
- At time $t = 20$ the schedule repeats, and the system is feasible.
The rest of the paper is organized thus: we shall first detail our context and how to use it; we shall then, as a first step, determine the first response time of each task in a synchronous system; next the general case will be considered; and finally we shall analyze the performances of the process.

2. Priority, Extended Schedules and Feasibility Intervals

The definition of DDS is a bit ambiguous; indeed, while two requests of the same task never have the same priority (they always have different deadline instants), the $k^{th}$ request of $\tau_i$ (which we shall denote $\delta_i^k$) and the $h^{th}$ request of $\tau_j$ (i.e., $\delta_j^h$) with $i \neq j$ may have the same priority, if $O_i + (k-1) \cdot T_i + D_i = O_j + (h-1) \cdot T_j + D_j$, and they may be active simultaneously (let us recall that in the static case two requests of the same task always have the same priority, but they are not very often simultaneously active; and two different tasks may have the same priority with RMS or DMS, if they have the same period or deadline delay).

This has no impact on the schedulability of the system: whatever way the ties are broken, even in ways that induce aperiodic schedules [3, 5], either all the deadlines are met or a first deadline failure occurs at the very same instant. However, the way to lift ambiguities has a definite impact on the response time of the various requests, i.e., the delay between the arrival of the request and its completion. We shall assume here that in case of identical deadline instants the priority is given to the request of the task with the smallest index; in summary, we shall have that $\delta_i^k$ has a higher priority than $\delta_j^h$, noted $\delta_i^k > \delta_j^h$, if $O_i + (k-1) \cdot T_i + D_i < O_j + (h-1) \cdot T_j + D_j$ or if $O_i + (k-1) \cdot T_i + D_i = O_j + (h-1) \cdot T_j + D_j$ and $i < j$; the developments presented in the rest of the paper may easily be adapted to cope with other tie breaking rules, provided that the relative priority of two requests does not change with time, which is not too strong a restriction$^1$.

We shall be interested in computing the response time of the $k^{th}$ request of $\tau_i$. In general, we shall assume that all the requests with a deadline before the considered completion time respect their deadline. However, our formulas will also work if the deadlines are soft and a request missing its deadline continues to work till its completion with the priorities given by DDS; we shall call this an extended schedule. On the contrary, they will not be valid with other rules specifying what happens when a deadline is missed, like the usual hard rule stopping the system at the first deadline miss, or the soft rule killing the requests when they miss their deadline but leaving the system to continue. Similarly, they will not hold for the kind of partially extended schedules which are sometimes considered for general deadline systems, where a request is allowed to pursue till the occurrence of the next request (at which time the request is dropped if not completed, in order to maintain the property that no two requests of a same task are active simultaneously). In all those cases, as for LLF schedules, it seems necessary to perform a true simulation of the system evolution in order to obtain the response times. Notice also that

$^1$It is not common to break a tie between requests $\delta_i^k$ and $\delta_j^h$ by giving the priority to task $\tau_i$ in even slot times and to task $\tau_j$ in odd slot times, for instance.
in those extended schedules, even if $U > 1$, the response time of any request will be finite (but
the successive response times will grow indefinitely when time passes), provided all deadlines are
finite (which we shall assume), since there are always only finitely many requests with a higher
priority than the considered one. Finally we may observe that with the extended schedules the
deadline of an active request may be far away in the past: that is why we specified that DDS
chooses the request with the least deadline and not the closest one, as is usually said.

In order to check the feasibility of a system, it is of course not possible to simulate the
system till the end of time, or equivalently to compute the response times of all the requests
of all the tasks. Fortunately, it is generally possible to determine a feasibility interval, i.e., an
finite interval such that if we only consider the requests occurring in this interval and if all the
deadlines in it are met, then all the deadlines will always be met. For instance, for synchronous
systems with arbitrary deadlines (thus including late and general deadline cases), the interval
$[0, L)$ is a feasibility interval, where $L$ is the first idle point, i.e., the first instant $t > 0$ such that
all requests occurring strictly before $t$ are completed at $t$ [3, 5]; if $U \leq 1$ (otherwise, we know that
the system is unfeasible), $L \leq P$, where $P$ is the least common multiple of the various periods
$T_i$. For asynchronous systems with arbitrary deadlines and $U \leq 1$, the interval $[R^{\text{min}}, O' + 2 \cdot P)$
is a feasibility interval, where $O' \geq O^{\text{max}} = \max\{O_i \mid i = 1, \ldots, n\}$ is some instant after the
largest offset and $R^{\text{min}} = \min\{O_i + T_i \cdot \left(\frac{O' - O_i}{P_i}\right) \mid i = 1, \ldots, n\}$ is the smallest occurrence time
for a request just before $O'$ [5]. Starting from $R^{\text{min}}$ instead of from 0 amounts to changing the
offsets in such a way that they do not differ too much. Unfortunately, in general $L$ as well as
$P$ grow exponentially with the number of tasks. Nevertheless those checks may be done if we
consider systems where those values are not too large (which may occur even for very large task
sets, by chance or because the user has some influence on the choice of the various periods) and
if we have quick enough procedures to compute the response times in those intervals, which is
the subject of this paper.

3. First Response Time in Synchronous Systems

We shall first consider the simplified case of the response time of the first request of a task in the
synchronous case, thus extending Audsley and Tindell's analysis to DDS. As usual, this response
time is the smallest value $r_i^1$ such that $r_i^1$ is exactly equal to the total interference from higher
priority requests, plus the computation due to $T_i$. In the rest of this paper, we shall often use
the notation $x^+ = \max\{x, 0\}$ in order to avoid the occurrence of negative values.

**Theorem 3.1.** Let $\gamma = \{\tau_1, \ldots, \tau_n\}$ be a synchronous task set with arbitrary deadlines; for our
deadline driven scheduler, $r_i^1$ is the smallest solution (if it is not greater than $D_i$ and the higher
priority requests meet their deadline too, or if we adopt the extended schedule) of the equation:

$$r_i^1 = C_i + \sum_{j < i} W_j^1(r_i^1) \cdot C_j + \sum_{j > i} w_j^1(r_i^1) \cdot C_j$$  \hspace{1cm} (1)
where

\[
W_j^i(\omega) = \min \left\{ \left[ \frac{(D_i + T_j - D_j)^+}{T_j} \right], \left[ \frac{\omega}{T_j} \right] \right\}, \quad (2)
\]

\[
w_j^i(\omega) = \min \left\{ \left[ \frac{(D_i - D_j)^+}{T_j} \right], \left[ \frac{\omega}{T_j} \right] \right\}. \quad (3)
\]

**Proof:**

We assume that either all requests of higher priority than \( \delta_j^i \) meet their deadline and \( r_j^i \leq D_i \), or the extended schedule is considered. We compute the interference of all higher priority requests in the interval \([0, r_j^i]\). We distinguish between the requests with a lesser index \((j < i, \text{case } (a))\) and those with a greater index \((j > i, \text{case } (b))\). It may be noticed that the other requests of \( \tau_i \), \( \text{i.e.}, \delta_k^i \text{ with } k > 1 \) have a lower priority than \( \delta_j^i \) and do not interfere.

(a) \( W_j^i(\omega) \) is the number of requests of \( \tau_j \) with a higher priority than the first request of \( \tau_i \) \((j < i)\), hence with a deadline less than or equal to \( D_i \), which occur strictly before \( \omega \), hence in the interval \([0, \omega)\). Its interest arises from the remark that, if \( \omega \leq r_j^i \), each of those requests must use \( C_j \) time units before the completion of the first request of \( \tau_i \).

It is easy to see that \( \left[ \frac{\omega}{T_j} \right] \) denotes the number of requests of \( \tau_j \) in the interval \([0, \omega)\), and 

\[
k = \left[ \frac{(D_i + T_j - D_j)^+}{T_j} \right]
\]

denotes the number of all requests of \( \tau_j \) in the interval \([0, D_i)\) with a deadline less than or equal to \( D_i \). Indeed \( k = 0 \) if \( D_j > D_i \), otherwise it is the largest integer such that \((k - 1) \cdot T_j + D_j \leq D_i \) (in which case \( k \geq 1 \)). Formula (2) follows.

(b) \( w_j^i(\omega) \) is defined similarly but here, since \( j > i \), we only have to consider the requests of \( \tau_j \) with a deadline strictly less than \( D_j \). It may be checked that 

\[
k = \left[ \frac{(D_i - D_j)^+}{T_j} \right]
\]

denotes the number of all requests of \( \tau_j \) in the interval \([0, D_i)\) with a deadline strictly less than \( D_i \), \( \text{i.e., such that } (k - 1) \cdot T_j + D_j < D_i \). Formula (3) follows.

If the interval \( r_j^i \) is equal to the interference of higher requests than \( \delta_j^i \) plus the computation time of \( \tau_i \) \( (C_j) \) that means that the interval \([0, r_j^i]\) is fully used and that \( \delta_j^i \) can complete its execution. The response time of \( \delta_j^i \) is the first such instant.

It may be observed that, even if the system is not feasible (e.g., if \( U > 1 \)), \( r_j^i \) is upper bounded in the extended schedule\(^3\) since so are \( W_j^i \) and \( w_j^i \), consequently

\[
r_j^i \leq B_i = C_i + \sum_{j < i} \left[ \frac{(D_i + T_j - D_j)^+}{T_j} \right] \cdot C_j + \sum_{j > i} \left[ \frac{(D_i - D_j)^+}{T_j} \right] \cdot C_j.
\]

\(^3\)Notice that this is not true for the other behaviour policies we mentioned in case of deadline failures, since then a higher priority request of \( \tau_j \) may be dropped before its completion. Moreover, this kind of reasoning may not be adapted to a LLF schedule since the relative priority between two requests may change during time and consequently the first request of \( \tau_j \) may be delayed by a fraction only of some request of \( \tau_j \).

\(^3\)This is not the true for static priority schedulers: if \( C_1 \geq T_1 \) and \( \tau_1 \) has the highest priority, \( r_j^i = \infty \).
Notice also that \( r^1_i \) occurs on both sides of the Equation (1): it is not given by an explicit formula. As a consequence, Equation (1) may have several solutions; for example, if we consider the synchronous system composed of two tasks \( \tau_1 \) and \( \tau_2 \) with \( \tau_1 = \{ C_1 = 1, D_1 = T_1 = 2, O_1 = 0 \} \) and \( \tau_2 = \{ C_2 = 1, D_2 = T_2 = 4, O_2 = 0 \} \), Equation (1) has 2 solutions: \( r^1_2 = 2 \) and \( r^1_2 = 3 \) (see Figure 2); only the first one is relevant.

The minimal value for \( r^1_i \) can be found by iteration:

\[
\begin{cases}
  z_0 = C_i, \\
  z_{k+1} = C_i + \sum_{j<i} W^i_j(z_k) \cdot C_j + \sum_{j>i} w^i_j(z_k) \cdot C_j.
\end{cases}
\]

The iteration is growing (since so are \( W^i_j \) and \( w^i_j \)) and proceeds until \( z_{k+1} = z_k = r^1_i \) (then we have a solution and it is not difficult to see that it is the smallest one). If we do not consider the extended schedules, the iteration may also be stopped if \( z_k \) exceeds \( D_i \) because \( \tau_i \) is then detected unschedulable. Since at each non-terminating iteration \( z_k \) is increased by some \( C_j \), the maximal number of iterations is \( \lfloor \frac{B-C_i}{\min_{j>i} C_j} \rfloor + 1 \), and \( \lfloor \frac{D-C_i}{\min_{j<i} C_j} \rfloor + 1 \) in the non-extended case. It is not difficult to see that this maximal number of iterations is very pessimistic: in practice the number of iterations is by far lower since the iterative process can stop with \( z_k < D_i \) and \( z_k \) can be increased by several \( C_j \)'s (and not necessarily the minimal one). We shall see experimental results concerning the exact number of iterations in section 5, for more general cases of response time computations.

From a schedulability point of view, the response time of the first request of \( \tau_i \) in the synchronous case is less instructive than the one for static schedulers with general or late deadlines, since it is not always the worst one: we have considered first this case for its potential simplifications and for didactic purposes, not for its practical interest.

## 4. General Response Time Computation

We shall now consider the more general response time computation: the response time \( \rho^k_i \) of the \( k^{th} \) request \( \delta^k_i \) of task \( \tau_i \) in asynchronous and arbitrary deadline systems (hence including synchronous, late and general cases) for our dynamic deadline driven scheduler. As in the previous section, this response time will be determined by the interference of higher priority requests and by the execution time of the considered request. The main differences arise from
the facts that the formulas have to take care of the possibility for a task to start its work after the arrival or even after the completion of the considered request; that the delay induced by a higher priority request may only be a fraction of its execution time (if it started to work but was not completed at the considered arrival time); and that an impact should not be accounted twice. This leads to the following result.

**Theorem 4.1.** Let γ = {τ₁, ..., τₙ} be an asynchronous task set with arbitrary deadlines; for our deadline driven scheduler, pkᵢ is the smallest solution of the equation:

\[ pkᵢ = Cᵢ + \pi + \sum_{j \neq i} n_j(pkᵢ) \cdot C_j \]  

with

\[ \pi = \max \{ \pi_j | j = 1, \ldots, n \} \]

\[ n_j(pkᵢ) = \begin{cases} \min \left\{ \left[ \frac{Rᵢ^k - Oᵢ}{T_j} \right], \left[ \frac{Rᵢ^k + Dᵢ + Oᵢ - D_j}{T_j} \right] \right\} & \text{if } j < i \\ \min \left\{ \left[ \frac{Rᵢ^k + Dᵢ - Rᵢ^k - T_j}{T_j} \right], \left[ \frac{Rᵢ^k + Dᵢ - Rᵢ^k}{T_j} \right] \right\} & \text{if } j > i \end{cases} \]

where

\[ Rᵢ^k = Oᵢ + (p - 1) \cdot T_j \]

\[ m_j = \begin{cases} \min \left\{ \left[ \frac{Rᵢ^k - Oᵢ}{T_j} \right], \left[ \frac{Rᵢ^k + Dᵢ + Oᵢ - D_j}{T_j} \right] \right\} & \text{if } (j < i) \land (Oᵢ < Rᵢ^k) \\ \min \left\{ \left[ \frac{Rᵢ^k - Oᵢ}{T_j} \right], \left[ \frac{Rᵢ^k + Dᵢ - Oᵢ - D_j}{T_j} \right] \right\} & \text{if } (j \geq i) \land (Oᵢ < Rᵢ^k) \\ 0 & \text{otherwise, i.e., if } Oᵢ \geq Rᵢ^k \end{cases} \]

\[ \pi_j = \begin{cases} (Rᵢ^m_j + D_j - Rᵢ^k)^+ & \text{if } m_j > 0 \\ (\wedge Rᵢ^m_j + D_j > Rᵢ^k \text{ if feasibility is assumed}) & \text{otherwise} \end{cases} \]

\[ z \text{ is such that } \pi_z = \pi \]

\[ m_j = \begin{cases} \left[ \frac{Rᵢ^k - Oᵢ}{T_j} \right] + 1 & \text{if } \pi = 0 \\ \max \left\{ \left[ \frac{Rᵢ^k - Oᵢ}{T_j} \right], \left[ \frac{Rᵢ^k + Dᵢ - Oᵢ - D_j}{T_j} \right] \right\} & \text{if } \pi > 0 \land j \leq z \\ \min \left\{ \left[ \frac{Rᵢ^k + Dᵢ - Oᵢ}{T_j} \right], \left[ \frac{Rᵢ^k + Dᵢ - Oᵢ - D_j}{T_j} \right] \right\} + 1 & \text{if } \pi > 0 \land j > z \end{cases} \]
Figure 3. Interference of requests which occur strictly before time $R_y^k$.

**Proof:**
As before, we may either assume that all higher priority requests than $\delta_i^k$ meet their deadlines, or consider the extended schedules with soft deadlines. $R_y^k$, as given by formula (7), denotes the arrival time of the $p^{th}$ request for $\tau_j$.

We may first observe that the higher priority requests occurring strictly before $R_y^k$, if any, may have a partial impact, in the sense that they may delay $\delta_i^k$ by a fraction of their computing time, if they started to use the CPU before $R_y^k$ but are not completed at that time. On the contrary, the requests occurring after $R_y^k$ either have no impact or delay $\delta_i^k$ by their whole computing time. Unfortunately, in order to know if a previous request is completed at $R_y^k$, one needs to determine its response time, so that the formula will generally be recursive (unless there is no previous higher priority request, or we know otherwise they are all completed). This will generally not be a problem since these formulas will normally be used to compute all the response times in some feasibility interval, by increasing arrival time or by decreasing priority (hence by increasing deadline instant), so that all the needed previous response times will be available.

Let us first consider those possible partial impacts. For a task $\tau_j$ (with $1 \leq j \leq n$), we only have to consider the last request (say the $m_j^{th}$) occurring strictly before $R_y^k$ (see Figure 3, where $D_y^k$ denotes the deadline of the $p^{th}$ request of $\tau_y$) with a higher priority than $\delta_i^k$ (if any, otherwise we take $m_j = 0$). Indeed, the interference of $\delta_j^r$ for $r < m_j$, if any, is included in the delay possibly created by $\rho_j^{m_j}$ since $R_j < R_j^{m_j}$ and $\delta_j^r > \delta_j^{m_j}$. If $O_j > R_y^k$, there is no such partial impact and $m_j = 0$, hence formula (8c). Let us thus assume that $O_j < R_y^k$ (which implies in particular that $k > 1$ if $i = j$). We also have $m_j = 0$ if $\delta_j^i > \delta_j^j$ since then the first request of $\tau_j$ already has too low a priority; this occurs if $O_j + D_j > R_y^k + D_i$ when $j < i$, and if $O_j + D_j \geq R_y^k + D_i$ when $j > i$ (it may not occur if $j = i$); this is compatible with the formulas (8a) and (8b) above. Otherwise, i.e., if $\delta_j^i > \delta_j^k$, $m_j = \max\{q(R_j^k < R_y^k) \text{ and } (\delta_j^k - \delta_j^i)\}$. The rank (say $x$) of the last request of $\tau_j$ which occurs strictly before time $R_y^k$ is $x = \left\lfloor \frac{R_y^k - O_j}{T_j} \right\rfloor$. Let $y$ be the rank of the last request of $\tau_j$ with a higher priority than $\delta_i^k$ (under our present hypotheses, it is at least 1); if $j < i$, $y$ is the largest integer such that $O_j + (y - 1) \cdot T_j + D_j \leq R_y^k + D_i$, i.e., $y = \left\lfloor \frac{R_y^k + D_i + T_j - O_j - D_i}{T_j} \right\rfloor$, and the formula (8a) follows; if $j \geq i$, $y$ is the largest integer such that $O_j + (y - 1) \cdot T_j + D_j < R_y^k + D_i$,
i.e., \( y = \left[ \frac{R_{x} + D_{x} - O_{j} - D_{j}}{T_{j}} \right] \), and the formula (8b) follows. Hence the formulas for \( m_{j} \).

The delay induced by the request \( \delta_{m_{j}, j} \), if \( m_{j} > 0 \), is equal to \( \pi_{j} = (R_{m_{j}}^{j} + \rho_{j}^{m_{j}} - R_{j}^{k})^{+} \) (which corresponds to the case (9a)), otherwise \( \pi_{j} = 0 \) (which corresponds to the case (9b)). We may avoid this computation if feasibility is assumed and \( R_{m_{j}}^{j} + D_{j} \leq R_{j}^{k} \), since then we know for sure that the request is completed and \( \pi_{j} = 0 \). Notice that the impact of \( \delta_{m_{j}, j} \) is not necessarily partial; it could happen that it did not use the CPU before \( R_{j}^{k} \); this is difficult to know however and does not invalidate our reasoning since we considered the last request possibly with a partial impact.

It may be observed that in arbitrary or soft deadline systems, it is not right to consider only the previous request of \( \tau_{j} \) (with respect to \( \delta_{j}^{k} \)) since this request may have a lower priority than \( \delta_{j}^{k} \) and consequently no impact on \( \rho_{j}^{k} \) while a previous request of \( \tau_{j} \) may be active at time \( R_{j}^{k} \) with a higher priority than \( \delta_{j}^{k} \) (this is the case in Figure 3: \( \rho_{j}^{m_{j}} \) has an impact on \( \rho_{j}^{k} \) while \( \rho_{j}^{m_{j}+1} \)

has no impact and \( R_{m_{j}}^{j} < R_{m_{j}+1}^{j} < R_{j}^{k} \)). Remark also that \( m_{i} = k - 1 \), and that the \( m_{i}^{th} \) request of task \( \tau_{i} \) may have an interference on \( \rho_{j}^{k} \); it follows that the term \( \pi_{i} \) must be considered.

Suppose now that two such requests of higher priority (say \( \delta_{a}^{m_{a}} \succ \delta_{b}^{m_{b}} \succ \delta_{j}^{k} \)) have an interference in the response time of the \( k \)th request of \( \tau_{i} \); in that case the interference of the higher priority one is included in the interference of the lower one: since at time \( R_{j}^{k} \) both requests are active, the request \( \delta_{a}^{m_{a}} \) ends its execution after the request \( \delta_{b}^{m_{b}} \), and the impact of the latter is included in the delay induced by the former. Hence, \( \pi_{a} > \pi_{b} > 0 \) and the total interference of all requests which precede strictly the \( k \)th request of \( \tau_{i} \) is equal to \( \pi = \max\{ \pi_{j}\mid j = 1, \ldots, n \} \). If this term is nonzero, we shall denote by \( z \) the (unique) index of the task realising it, i.e., such that \( \pi_{z} = \pi \). Hence the formulas for \( \pi \) (5) and \( z \) (10).

Let us now consider the requests of \( \tau_{j} \) from the first one, say the \( \tilde{m}_{j} \), whose impact is not included in the term \( \pi \) (this does not mean that it will have a true impact on \( \rho_{j}^{k} \); this will be considered next). We have \( R_{m_{j}}^{j} \geq R_{j}^{k} \) since a request occurring strictly before \( R_{j}^{k} \) either is completed before \( R_{j}^{k} \) (and may be considered as included in \( \pi \) but with a null impact) or is included in the term \( \pi \), or has a lower priority than \( \delta_{j}^{k} \) (and may also be considered as included in \( \pi \) but with a null impact). We may assume \( j \neq i \) since the next request of \( \tau_{j} \) from \( R_{j}^{k} \) is \( \delta_{j}^{k} \) itself, and the following ones have a lower priority. If \( \pi = 0 \), there is no other constraint and \( \tilde{m}_{j} = x_{1} = \left[ \frac{(O_{j}^{k} - O_{j})^{+}}{T_{j}} \right] + 1 \) is the rank of the first request of \( \tau_{j} \) after or at \( R_{j}^{k} \), justifying the formula (11a). If \( \pi \neq 0 \), \( \tilde{m}_{j} \leq x_{2} = \left[ \frac{(O_{j}^{k} + n - O_{j})^{+}}{T_{j}} \right] + 1 \), the rank of the first request of \( \tau_{j} \) after or at \( R_{j}^{k} + \pi \). But if between \( \delta_{j}^{r_{1}} \) and \( \delta_{j}^{r_{2}} \) there are requests with a lower priority than \( \delta_{j}^{m_{z}} \), they are not included in \( \pi \) and must also be considered. Let \( x_{3} \) be the rank of the first request of \( \tau_{j} \) with a priority strictly lower than \( \delta_{j}^{m_{z}} \): if \( j \leq z \), \( x_{3} \) is the least natural (from 1) such that \( O_{j} + (x_{3} - 1) \cdot T_{j} + D_{j} > R_{m_{z}}^{j} + D_{z} \), so that \( x_{3} = \left[ \frac{(R_{m_{z}}^{j} + D_{z} - O_{j} - D_{j})^{+}}{T_{j}} \right] + 1 \); if \( j > z \), \( x_{3} \) is the least natural (from 1) such that \( O_{j} + (x_{3} - 1) \cdot T_{j} + D_{j} \geq R_{m_{z}}^{j} + D_{z} \), so that \( x_{3} = \left[ \frac{(R_{m_{z}}^{j} + D_{z} - O_{j} - D_{j})^{+}}{T_{j}} \right] + 1 \). Hence, \( \tilde{m}_{j} = \max\{ x_{1}, \min\{ x_{2}, x_{3} \} \} \), justifying the formulas (11b) and (11c). Hence the formula for \( \tilde{m}_{j} \).

Now, \( n_{j}(\rho_{j}^{k}) \) denotes the number of higher priority requests (with respect to \( \delta_{j}^{k} \)) of \( \tau_{j} \) which
occur in the interval $[R^{m_j}_j, R^k_i + \rho^k_i]$. From our hypotheses, for each of them the response time \( \rho^k_i \) undergoes a delay of \( C_j \) time units, since the corresponding request has to be fully executed meanwhile. The number of requests of \( \tau_j \) in this interval is 
\[
x = \left\lfloor \frac{(R^k_i + \rho^k_i - R^{m_j}_j) +}{T_j} \right\rfloor. \tag{7a}
\]
Let \( x' \) be the number of requests of \( \tau_j \) from \( R^{m_j}_j \) with a higher priority than \( \delta^k_i \). If \( j < i \), \( x' \) is the largest natural integer (from 0) such that 
\[
R^{m_j}_j + (x' - 1) \cdot T_j + D_j \leq R^k_i + D_i, \tag{7b}
\]
hence \( x' = \left\lfloor \frac{(R^k_i + D_i - R^{m_j}_j - T_j) +}{T_j} \right\rfloor \). And \( n_j(\rho^k_i) \) is the least of these two numbers \( x \) and \( x' \), as given by the formulas (6a) and (6b).

We have thus shown that the interference of higher priority tasks than \( \tau_i \) in the interval $[R^k_i, R^k_i + \rho^k_i]$ plus the computation of \( \tau_i \) is 
\[
I(\rho^k_i) = C_i + \pi + \sum_{j \neq i} n_j(\rho^k_i) \cdot C_j
\]
and the \( k^{th} \) request of \( \tau_i \) ends its computation at the first instant \( R^k_i + \rho^k_i \) such that the equality 
\( \rho^k_i = I(\rho^k_i) \) is satisfied. \( \square \)

The effective computation of \( \rho^k_i \) can be divided into two parts:

- the computation of the term \( \pi \) (and the related numbers: \( m_j, \rho^m_j, z, m_j \)), which is immediate if we assume that the needed previous response times have been computed before, which we did,

- and the computation of the least solution of the equation:

\[
\rho^k_i = C_i + \pi + \sum_{j \neq i} n_j(\rho^k_i) \cdot C_j.
\]

which can again be resolved by an iterative process:

\[
w_0 = C_i + \pi,
\]
\[
w_{k+1} = C_i + \pi + \sum_{j \neq i} n_j(w_k) \cdot C_j,
\]
till stabilisation.

If we are only interested in the feasibility of the system, we may stop the iterations before the stabilisation, if \( w_k \) exceeds \( D_i \). Since at each nonterminating step one or more \( C_j \)'s are added to \( w_k \), that means that the number of iterations is bounded by 
\[
\left\lfloor \frac{(D_i - C_i - \pi - \min_{j \neq i} \frac{C_j}{C_j})}{\min_{j \neq i} C_j} \right\rfloor + 1.
\]
If we are interested in extended schedules and $U \leq 1$, the system is finally periodic (see for instance [5]) and we know that all the response times are bounded, hence the iterations always stabilise. If $U > 1$, the term $\pi$ indefinitely grows with $k$, as exhibited for instance by the very simple system $S = \{\tau_1 = \{T_1 = 2, C_1 = 3, O_1 = 0\}\}$: the terms $\pi$ in the computation of $\rho_k^i$ (for increasing values of $k$) are $0, 1, 2, \ldots$ As a consequence, $\rho_k^i$ also grows indefinitely with $k$. But it remains finite, together with $\pi$, since each $n_j(\rho_k^i)$ is upper bounded by

$$\left[\frac{(R_k^i + D_i - R_{ij}^k + T_j - D_j)^+}{T_j}\right] \leq \left[\frac{(D_i - D_j)^+}{T_j}\right]$$

depending on $j$ being lower than $i$ or not, and in any case this is independent of $\rho_k^i$. The second bounds arise from the fact that $R_k^i \leq R_{ij}^k$, and they deserve a special interest since they are even independent of $k$. Hence, we have the bounds

$$\rho_k^i \leq C_i + \pi + \sum_{j<i} \left[\frac{(R_k^i + D_i - R_{ij}^k + T_j - D_j)^+}{T_j}\right] \cdot C_j + \sum_{j>i} \left[\frac{(R_k^i + D_i - R_{ij}^k - D_j)^+}{T_j}\right] \cdot C_j$$

$$\leq C_i + \pi + \sum_{j<i} \left[\frac{(D_i - D_j)^+}{T_j}\right] \cdot C_j + \sum_{j>i} \left[\frac{(D_i - D_j)^+}{T_j}\right] \cdot C_j.$$}

As a consequence, not only the iterative process always stabilises, but the number of iterations is bounded by $\left[\frac{\Delta_i}{\min_{j \neq i} C_j}\right] + 1$, with $\Delta_i = \sum_{j<i} \left[\frac{(D_i - D_j)^+}{T_j}\right] \cdot C_j + \sum_{j>i} \left[\frac{(D_i - D_j)^+}{T_j}\right] \cdot C_j$.

It could be objected that the procedure just described is nothing more than a simulation in disguise, and in some sense this is true. But by exploiting the characteristics of the deadline driven scheduler and of the kind of extended schedules we considered here, we have turned it into a simple computation process, by far much faster than a brute force simulation. On the contrary, we have not found how to translate this kind of computation to other extended schedules, nor to the LLL schedulers, nor to the case of many processors.

The formulas above, while being rather simple to evaluate, may seem a bit complex; they may be slightly simplified in the case of general deadlines if we assume that they are used in a feasibility test and no deadline miss was observed before.

**Theorem 4.2.** Let $\gamma = \{\tau_1, \ldots, \tau_n\}$ be an asynchronous task set with general deadlines; for our deadline driven scheduler, if no deadline was missed before, the formulas for $\pi_j$ and $\hat{m}_j$ may be simplified in

$$l_j = \left[\frac{R_k^i - O_j}{T_j}\right] \quad \text{if } O_j < R_k^i$$

$$\pi_j = \begin{cases} (R_{ij}^k + \rho_j^i - R_k^i)^+ & \text{if } j \neq i \text{ and } O_j < R_k^i < R_{ij}^k + D_j \text{ and } \delta_j^i \geq \delta_k^i \\ 0 & \text{otherwise} \end{cases}$$

(12)
\[ m_j = \left\lceil \frac{(R_i^k + \pi - O_j)^+}{T_j} \right\rceil \] (14)

\[ \tilde{m}_j = \begin{cases} m_j & \text{if } m_j > 0 \text{ and } R_j^i \geq R_i^k \text{ and } \delta_j^i \leq \delta_z^m \text{ and } \\
 m_j + 1 & \text{otherwise} \end{cases} \] (15a) (15b)

**Proof:**

The general argument is as in theorem 4.1. We shall only stress here the differences justifying the mentioned simplifications.

For a task \( \tau_j \) (with \( j \neq i \)), we only have to consider the requests from the \( l_j^i \)th, the last one that precedes strictly \( R_i^k \) (see Figure 4) if any, i.e., if \( O_j < R_i^k \), since from our hypotheses the previous requests are completed and do not have any direct\(^4\) impact on \( \rho_i^k \). For the same reason, we do not have to consider the requests of \( \tau_i \) before the \( k^i \)th, so that we may assume \( j \neq i \). Hence, a nonzero \( \pi_j \) may only occur if \( j \neq i \), \( O_j < R_i^k < \tilde{R}_j^i + D_j \) and \( \delta_j^i > \delta_k^i \), justifying formulas (13a) and (13b).

Let us consider now the interference of requests which occur after or at time \( R_i^k \). For each task \( \tau_j \) (with \( j \neq i \)) we have to consider the requests from the \( m_j^i \)th, i.e., the first one which is not included in the term \( \pi \). Let \( \hat{m}_j \) denote the rank of the last request of \( \tau_j \) which occurs strictly before \( R_i^k + \pi \) (\( \hat{m}_j = 0 \) if no request occurs before this time); it can be seen that \( \hat{m}_j = \left\lceil \frac{(R_i^k + \pi - O_j)^+}{T_j} \right\rceil \), justifying the formula (14). We may distinguish four cases: (i) \( \hat{m}_j = 0 \), (ii) \( R_j^i < R_i^k \), (iii) the request \( \delta_j^i \) has a higher priority than \( \delta_z^m \), or is \( \delta_z^m \), and (iv) the request \( \delta_j^i \) has a lower priority than \( \delta_z^m \).

(i) If \( \hat{m}_j = 0 \), i.e., if task \( \tau_j \) does not start before \( R_i^k + \pi \), we have to consider the requests from the first one and consequently \( \hat{m}_j = \hat{m}_j + 1 = 1 \), which corresponds to the case (15b).

(ii) Otherwise, if \( R_j^i < R_i^k \), by definition the request \( \delta_j^i \) was taken into account in the term \( \pi \) (this is the case of the request \( \delta_j^i \) in Figure 5); hence, we have to consider the requests of \( \tau_j \) from the rank \( \hat{m}_j + 1 \), i.e., from the first one which occurs after or at \( R_i^k + \pi \), which corresponds again to the case (15b); notice that this will always be the case if \( \pi = 0 \).

\(^4\)But they may have an indirect interference, by delaying a request with priority between it and \( \delta_k^i \) which is not finished at \( R_i^k \).
(iii) Otherwise, if $\delta_{j}^{\tilde{m}_{j}} \geq \delta_{z}^{\tilde{m}_{z}}$, all requests of $\tau_{j}$ up to $\delta_{j}^{\tilde{m}_{j}}$ have a higher priority than $\delta_{z}^{\tilde{m}_{z}}$, or is $\delta_{z}^{\tilde{m}_{z}}$, and were already considered in the term $\pi$; hence we have to consider the requests from the next one, i.e., $\tilde{m}_{j} = \tilde{m}_{j} + 1$ (this is the case of the request $\delta_{j}^{\tilde{m}_{j}}$ in Figure 5), which corresponds again to the case (15b).

(iv) Otherwise, $\delta_{j}^{\tilde{m}_{j}} < \delta_{z}^{\tilde{m}_{z}}$ and the interference of this request is not included in the term $\pi$ (this is the case of the request $\delta_{l}^{\tilde{m}_{l}}$ in Figure 5); on the contrary, the previous request has a deadline before $R_{j}^{\tilde{m}_{j}}$, hence before $R_{j}^{l} + \pi = R_{z}^{\tilde{m}_{z}} + \rho_{z}^{\tilde{m}_{z}}$, and also before the deadline of $\delta_{j}^{\tilde{m}_{j}}$ since we assumed that all the previous requests met their deadline; as a consequence, $\delta_{j}^{\tilde{m}_{j} - 1}$ has a higher priority than $\delta_{z}^{\tilde{m}_{z}}$ and was considered in $\pi$; we must thus consider the requests of $\tau_{j}$ from rank $\tilde{m}_{j} = \tilde{m}_{j}$. This is the only case where $\tilde{m}_{j} \neq \tilde{m}_{j} + 1$, which corresponds to the case (15a).

The effective computation of the response times in this case may be performed with the same kind of iterative procedure as before.

5. Performance analysis

We have given above various bounds on the number of iterations needed to compute the smallest solution of the response time equation in Theorem 4.1; however, these bounds are very pessimistic; their main justification was to show that the procedure not only converges, but also that it does not degrade with time (since some bounds do not rely on the rank of the computed response time). In order to get a better grasp on the actual performances, we have implemented the iterative process above and we have compared the actual number $a$ of iterations with the bound $b = \lceil \frac{D_{l} - C_{l} - \pi}{\min_{j} \bar{c}_{j}} \rceil + 1$. We have applied this experimentation on randomly chosen task
sets; \( n \) was chosen randomly in the interval \([20, 100]\), the periods \( T_i \) in the interval \([50, 1000]\), the deadlines \( D_i \) in the interval \([\frac{T_i}{2}, T_i]\), the offsets \( O_i \) in the interval \([0, T_i]\) and the computation times \( C_i \) in order to have a large utilization factor (i.e., near 1).

Figures 6 and 7 show first the frequency of the actual number of iterations and those given by the bound, respectively, for the restricted variation domain \( n \in [50, 60] \) in order not to incur the danger of mixing up too different phenomena. Figure 6 shows that the main part of the probability mass of the actual number is situated in \([0, 10]\). Figure 7 shows that the number of iterations given by the bound is significantly larger: the main part of the probability mass is situated in \([25, 1500]\).

Figure 8 shows the distribution of the ratio \( \frac{r}{n} \) and exhibits that the main part of the probability mass is less than 0.01; there are very few individuals for which the ratio is greater than 0.02.

We have observed the same kind of behaviour (such as illustrated by Figures 6 to 8) for the other variation domains of \( n \) in our simulations.

Figure 9 then exhibits the average ratio in function of the system size: it may be noticed that it does not vary much with the number \( n \) of tasks, and is around 0.8 %, hence the claimed pessimistic nature of the bound and the relative efficiency of the iterative process.

We have not distinguished in this study feasible and unfeasible systems, nor the behaviour of the higher bound to be used for extended schedules: this remains open for further research.

Let us now turn to the space complexity of the procedure. We assumed that all the response
Figure 7. Frequency of the number of iterations given by the bound.

Figure 8. Frequency of the ratio $\frac{a}{b}$.
times of the requests occurring before $R_k^c$ were known, hence stored. This may be quite large since the feasibility intervals to be considered have a length which may be exponential in the number of tasks, so that the number of response times to store may also be exponential.

Indeed, the general feasibility interval $[R_{\text{min}}^c, O' + 2 \cdot P]$ mentioned in section 2 has an approximate length of $2 \cdot P$, and $P$ may grow very fast. In order to illustrate this, we determined the distribution of $P$ for 20 tasks with periods randomly chosen in $[50, 100]$. Figure 10 (see also [11]) shows that the interval length in that case is usually in the billion range.

The situation is slightly better in the synchronous case, where the length of a feasibility interval is $L$, the position of the first idle point. We said that the latter is bounded by $P$, but in general it is by far lower, except when the utilization factor is close to 1. In order to illustrate the fact that the bound $P$ is very pessimistic for $L$, we shall consider these values on randomly chosen late deadline task sets for $n = 20$, the periods are randomly chosen in $[50, 100]$, and the computation times are chosen in order to have utilization factors in the interval $[0.3, 1]$; the systems are not necessarily feasible but the extended schedules are considered. Figure 11 shows the distribution of $L$ for systems where the utilization factor $U \in [0.85, 0.95]$. Figure 12 shows the distribution of the ratio $\frac{L}{P}$ (it may be noticed that the minimal value for $P$ in our simulations is near $10^4$; we have observed the same behaviour for other variation domains of $U$) and Figure 13 shows the average ratio $\frac{L}{P}$ in function of the utilization factor: it increases with $U$ but remains less than $7 \cdot 10^{-6}$ (for $U \leq 0.95$), exhibiting the fact that the bound $P$ is very pessimistic for $L$, even if $U$ is near 1. It could be also interesting to study the ratio $\frac{L}{P}$ in the close neighborhood of 1, but this question remains for further research.

Fortunately, in practical applications, it will sometimes be possible to act on the choice of the
Figure 10. Frequency of the value $P$.

Figure 11. Frequency of the value $L$. 
Figure 12. Frequency of the ratio $\frac{f}{r}$.

Figure 13. The average ratio $\frac{L}{p}$ in function of $U$. 
periods in order to limit the values of $P$ (and $L$, consequently). Moreover, it is not necessary to
store all the previously computed response time in order to apply our procedure: only active ones
at some instants close to $R_i^k$ are needed. And if the system was feasible up to the computation
of $\rho_i^k$, their number is bounded.

**Theorem 5.1.** The maximal space complexity of the computation of $\rho_i^k$ in a feasibility check
is $O(n \cdot \left\lceil \frac{D_{\text{max}} - D_{\text{min}}}{T_{\text{min}}} \right\rceil)$, where $D_{\text{max}}$ and $D_{\text{min}}$
are the greatest and the lowest deadline delays, respectively, and $T_{\text{min}}$ is the least period.

**Proof:**
We shall show that we have to keep at most $\left\lceil \frac{D_{\text{max}} - D_{\text{min}}}{T_{\text{min}}} \right\rceil + 2$ response times for each task, in
order to be able to compute the various terms $\pi$.

For each pair of tasks $\tau_i$ and $\tau_j$ with $j < i$ (if $j \geq i$ the situation is similar), when we need to
determine $\rho_j^{m_j}$, $m_j$ is the rank of the last request of $\tau_j$ such that $R_j^{m_j} < R_i^k < R_j^{m_j} + D_j \leq R_i^k + D_i$, i.e., with a higher priority than $\delta_i^k$ and the possibility to overlap $R_i^k$.

As a consequence, $R_j^{m_j} + D_j + D_j > R_i^k + D_i$. And between time $R_j^{m_j}$ and $R_i^k$, there are at
most $\left\lceil \frac{R_i^k - R_j^{m_j}}{T_j} \right\rceil + 1 \leq \left\lceil \frac{D_j - D_i}{T_j} \right\rceil + 2$ requests of $\tau_j$, hence the claimed property. $\Box$

The situation is still more favourable in the case of general deadlines since then there is at
most one active request for each task at any time if the system is feasible.

6. **Conclusion**

In this paper, we have extended the computation of the response time previously devised for
the first request of synchronous tasks with a static scheduler to any request with the most
popular dynamic priority rule in the most general case. We have shown why it is interesting to
consider these response times regarding the feasibility problems of these more general systems.
We have also considered the problem of the effective computation of these response times for
interesting special cases. Finally, we have studied the analytical and experimental (time and
space) complexity of our algorithms. Of course more statistical analysis should be collected to
get a better understanding of their characteristics, but things seems to look quite agreeable.

**References**

[1] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with

elles, Belgium, 1999.


