INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Holistic Analysis for Deadline Scheduled Real-Time Distributed Systems

Marco Spuri

N° 2873

Avril 1996

THÈME 1

ISSN 0249-6399

INRIA
ROCQUENCOURT

Holistic Analysis for Deadline Scheduled Real-Time Distributed Systems

Marco Spuri*

Thème 1 — Réseaux et systèmes
Projet Reflecs

Rapport de recherche n°2873 — Avril 1996 — 31 pages

Abstract: The holistic theory is a very interesting approach formerly proposed by Tindell and Clark [23] for assessing the feasibility of fixed priority real-time systems. Its major merit is to make the analysis of distributed systems tractable, without being at the same time too pessimistic. In this paper we extend the holistic theory to the analysis of deadline scheduled real-time distributed systems.

Owing to its predictability, the Timed Token MAC protocol is assumed to arbitrate network accesses among host processors. Furthermore, in order to achieve a large resource utilization, outgoing packets are assumed to be locally queued earliest deadline first. A procedure for the computation of worst-case message communication delays is also given.

The theory described in the paper is validated by means of a case study application, in which worst-case response times of *end-to-end* computations are tightly bounded. The example has confirmed the effectiveness of a global deadline scheduling approach.

Key-words: real-time computing, deadline scheduling, feasibility analysis, distributed systems, Timed Token MAC protocol.

(Résumé : tsvp)

*This work has been supported by the Commission of the European Communities under contract ERBCHBGCT930458, proposal ERB4050PL931117.

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
Téléphone : (33 1) 39 63 55 11 – Télécopie : (33 1) 39 63 53 30

Analyse Holistique de Systèmes Répartis Temps-Réel à Ordonnancement par Échéance la plus Proche en Premier

Résumé : La théorie *holistique*, proposée par Tindell et Clark [23], est une approche très interessante pour établir la faisabilité de systèmes temps-réel à priorités fixes. Son principal mérite est de permettre l'analyse de systèmes répartis, sans que celle-ci soit trop pessimiste. Dans ce papier nous étendons la théorie holistique à l'analyse de systèmes répartis temps-réel ordonnancés selon les échéances.

Grâce à sa predictabilité, le protocole Timed Token a été consideré pour l'arbitrage des accès au réseau de la part des différentes stations. En plus, pour obtenir une bonne utilisation des ressources, nous avons supposé que les paquets émis sont localement ordonnancés par échéance la plus proche en premiér. Nous avons aussi developpé une procédure pour le calcul des délais de communication en pire cas.

La théorie décrite dans le papier a été validée sur un exemple d'application distribuée temps-réel, pour laquelle les pires temps de réponse des traitements de *bout-en-bout* sont strictement bornés. L'exemple a confirmé l'efficacité de l'approche à ordonnancement global par échéances.

Mots-clé : calcul et communications temps-réel, ordonnancement par échéance, analyse de faisabilité, systèmes répartis, protocole Timed Token.

1 Introduction

Control systems represent the natural application domain of real-time computing. Such systems have often a distributed architecture: data are collected by sensor devices, transmitted to processing devices, and actions are finally taken by actuator devices. That is, the behaviour of the system consists mainly in responding to trigger events whenever they are detected. In order not to jeopardize the process under control, these responses, which are the results of *end-to-end* computations, must be taken within maximum delays, termed *deadlines*, specified at design time.

Owing to the strictness of deadlines, real-time systems must be designed in such a way to give the a priori guarantee that the timing constraints are met even under peak load conditions. In order to achieve this design goal, suitable allocation techniques and predictable scheduling algorithms are needed in the processing resources as well as in the networking ones.

In its general formulation, the problem of assessing the feasibility of a real-time distributed system, that is, establishing whether all timing requirements are met, is NP-hard [22]. In order to overcome this inherent difficulty, problem restrictions and heuristics must be used. A common approach is to statically allocate application tasks at host processors [17, 16], and locally utilize either a well known scheduling algorithm like Rate Monotonic or Earliest Deadline First (EDF) [14], or a template schedule layed out at system design time [12]. Alternatively, a complete dynamic approach strongly based on heuristics can be followed [21].

In this paper we assume a system configuration in which application tasks are statically allocated to host processors and scheduled according to the EDF algorithm. The allocation may be the result of a suitable analysis and of hardware constraints. This issue is no longer addressed in what follows. The choice of the EDF algorithm is motivated by its predictability, as well as by its optimality in uniprocessor scheduling [20].

End-to-end deadline guarantees are possible only if the communication network supports the timely delivery of inter-task messages [2]. Among the communication protocols that provide this degree of predictability we find the Timed Token medium access control protocol [10], currently included in a number of local area network standards, namely the fiber distributed data interface (FDDI), the IEEE 802.4, the high speed data bus and the high speed ring bus (HSDB/HSRB), and the survivable adaptable fiber optic enbedded network (SAFENET) [2].

In the description of the following sections we assume network accesses are granted to host processors by using the Timed Token MAC protocol. Furthermore, in order to achieve a higher resource utilization, we also assume that outgoing packets are locally queued earliest deadline first. Note that this form of network scheduling has been recently suggested for the establishment of *real-time channels* [9, 27].

In order to analyse the feasibility of this sort of systems, we propose an innovative approach termed *holistic analysis*. The holistic approach has been formerly introduced by Tindell and Clark [23], who have described how to assess the feasibility of real-time distributed systems composed of fixed priority scheduled host processors and a time division

multiple access network. Later, Tindell *et al.* [24] have extended the analysis to systems with, respectively, a real-time and a timed token communication protocols.

In the holistic approach, end-to-end response times are tightly bounded and then compared to the design requirements, in order to establish whether the system is feasible. End-to-end bounds are computed by accurate analysing each subsystem (host processor or communication medium) in order to exactly evaluate worst-case task and message response times. Since the timing analysis of the messages is affected by the response times of the sender tasks, and similarly the timing analysis of the destination tasks is affected by the communication delays of the messages, the overall analysis of the different subsystems is repeated several times until stability is reached.

Note that one of the key aspects of the holistic approach is the ability to compute worst-case response times both for tasks and messages. A procedure for such a computation in the context of uniprocessor EDF scheduling has been described in [20] and will be briefly recalled in a later section. In this paper we propose a procedure to compute worst-case communication delays of inter-task messages, when the Timed Token MAC protocol is used to arbitrate accesses to the network, and when outgoing packets are locally queued at communications adapters earliest deadline first. Two subcases are considered: when both *synchronous* and *asynchronous* services are utilized by host processors, and when only the synchronous service is allowed.

The problem of estimating the average task response times in real-time distributed systems with resource contentions was tackled by Chu *et al.* in [7]. In [8] end-to-end guarantees are given by splitting the scheduling problem in two phases, one off-line, one on-line. The off-line scheduler builds a template by assigning each task an *execution window*, according to the precedence constraints and the resource needs. The local on-line schedulers then try to actually run non-preemptively the tasks in their assigned windows. The assumed architecture is based on a collection of VME-bus multiprocessors connected by a fiber optic reflective memory network.

Ramamritham [18] suggests the use of deadline driven heuristics for the allocation and the static scheduling of periodic tasks related by precedence, communication, and replication requirements. In the description a predictable protocol for network accesses, like point-to-point or TDMA, is assumed. Deadline driven heuristics have also been used by Bettati and Liu [5] for the solution of several *flow-shop* problems: timing constraints are given as end-to-end release times and deadlines, all the tasks execute on different processors in turn in the same order. In the Mars system [12] a time-triggered architecture is assumed. Everything is known before the system is run, and the schedule is statically layed out by means of heuristics. Access to the network is granted by using a TDMA protocol.

The rest of the paper is structured as follows. Section 2 describes the details of the assumed system model and the notation later utilized. The holistic approach is reviewed in Section 3. In Section 4 the results of [20] for the analysis of uniprocessor deadline scheduled systems is briefly recalled. The procedure for the timing analysis of the communication medium is described in Section 5. The computation of end-to-end response times is proposed in Section 6. The analysis presented throughout the paper is then applied to a case study

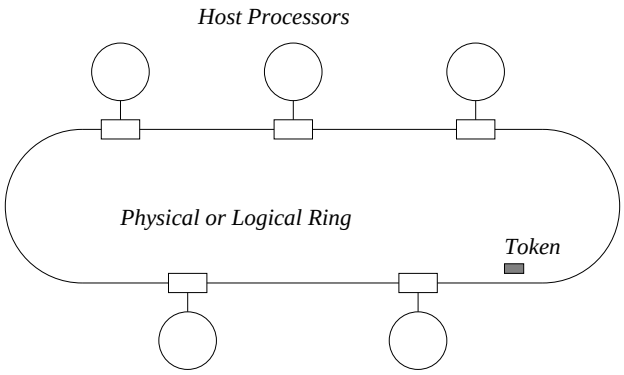

Figure 1: Assumed network topology.

example in Section 7. In Section 8 our last remarks are finally discussed and our conclusions stated.

2 System Model

The typical system we consider in our analysis is composed of several host processors, or nodes, connected by a physical or logical ring (in this case the actual network may be a shared broadcast bus), as shown in Figure 1.

In each station we find a set of statically allocated tasks, which possibly communicate over the network with other tasks on different processors. A *task* i consists of an infinite number of *requests*, or *instances*, whose *arrival* times are separated by a minimum time T_i, called *period* (according to the conventional notation, this assumption is common to periodic and sporadic tasks). We assume that task instances may arrive at any time. However, the arrival must be recognized by a run-time dispatcher, which then will place the instance in the system *ready* queue. The instance is then said to be *released*. Note that the release of a task can also be delayed by a distributed synchronization if the task is the destination of a message. The time between a task's arrival and its release is known as *release jitter*.

Each task instance may execute for a bounded amount of computation C_i, called *worst-case execution time*. The computation should complete within a time D_i (*relative deadline*) after the arrival. The ready queue is ordered according to the actual task's deadlines, earliest first, that is, we assume an EDF [14] pre-emptive dispatching. Tasks may also locally share resources, by locking and unlocking semaphores according to a protocol like the Priority Ceiling [19, 6] or the Stack Resource Policy [4].

Communicating tasks can send messages at any time, that is, as soon as they start executing, or as late as they complete. We assume that each message m, sent by task i, may be sent at most once every n_m invocations of i, and has a unique destination. Each task may receive at most one message.

When queued, messages may be broken down into a number of packets of fixed size (message m is assumed to be broken into C_m packets). The queue of the outgoing packets, locally shared between the host processor and the communications adapter, is ordered by increasing deadlines. How deadlines are assigned to messages will be described later.

Access to the ring, or the bus, is arbitrated by using the *Timed Token* medium access control protocol [10]. The Timed Token protocol normally provides two classes of service: the *synchronous* class and the *asynchronous* class [11]. The former class is intended for messages with regular arrivals and delivery time constraints. The latter class is intended for messages with arbitrary arrival laws and without time constraints. We assume that all messages involved in end-to-end computations with strict deadlines are serviced by the synchronous class.

Access control among the hosts is provided by a special bit pattern called *token* that circulates around the ring. At network initialization time, all hosts negotiate a common value for the *target token rotation time* ($TTRT$). Each host p is then assigned a fraction H_p of $TTRT$, termed *synchronous bandwidth*, and which is the maximum time the host can transmit synchronous messages upon reception of the token.

After the transmission of synchronous packets, if any, the station can send asynchronous messages only if it has received the token earlier than $TTRT$ units of time after the last token visit. The duration of the possible asynchronous message transmission is limited to $TTRT$ minus the time elapsed between the previous and the current visit. The token is immediately released after the transmission of the last packet.

Target token rotation time and synchronous bandwidths are related by the following inequality:

$$\sum_{p=1}^{n} H_p \ + \ \tau \ \leq \ TTRT,$$

where τ is the sum of protocol overheads and ring latency, that is, the fraction of $TTRT$ not available for message transmission.

In this paper we propose the analysis of the Timed Token MAC protocol in two different situations. In the first one we assume that full utilization of synchronous and asynchronous class services is allowed at any node. In the second one only the synchronous service is allowed, a situation that should give more responsiveness to the system.

Note that a number of papers concerning the time predictability of the Timed Token MAC protocol have appeared in the literature [15, 26, 2, 25]. However, the main concern of the authors has been usually to describe a good *synchronous bandwidth allocation* scheme, in order to guarantee the delivery constraints of message sets, whenever the *utilization factor* of the given set is less than the *worst-case achievable utilization*, which is a characteristic of the allocation scheme. The computation of worst-case message response times has been addressed by Tindell *et al.* [24] for a timed token protocol in which they have implicitly assumed to allow only the synchronous service at any node. Furthermore, outgoing packets are assumed to have fixed priorities and to be queued accordingly. A contribution we give in this paper is the description of a procedure for the computation of worst-case message

response times when message packets are locally queued earliest deadline first, and when either the full or the restricted version of the Timed Token protocol is utilized.

Owing to the similarity with the holistic approach originally described by Tindell and Clark in [23] and Tindell *et al.* in [24], in this paper we will try to use the same notation as far as possible. A glossary follows:

C_i The worst-case computation time of task i on each release.

D_i The deadline of task i, measured relative to the arrival time of the task.

B_i The worst-case blocking time task i can experience due to the operation of the concurrency control protocol.

J_i The worst-case release jitter of task i (*i.e.* the worst-case delay between the arrival and its release).

T_i The period of task i.

r_i The worst-case response time of task i, measured from the arrival time to the completion time.

C_m The number of packets message m is composed of.

D_m The deadline of message m, measured relative to its queuing time.

J_m The worst-case release jitter of message m (*i.e.* the difference between its possible latest and earliest queuing times).

T_m The period of message m.

r_m The worst-case communication delay of message m, measured from the time it is queued by its sending task, to the arrival at the destination processor.

ρ The time to transmit a packet.

H_p The *synchronous bandwidth* of node p, that is, the maximum time node p is allowed to transmit synchronous packets upon receipt of the token.

τ The ring latency and the protocol overheads. It is the fraction of $TTRT$ not available for message transmission.

$TTRT$ The target token rotation time. $TTRT \geq \tau + \sum_p H_p$. Note that in the restricted version of the Timed Token protocol we can assume the equality, and $TTRT$ becomes the worst-case token rotation time.

$\mathcal{P}$ The network propagation delay.

3 The Holistic Approach

The goal of this paper is primarily to describe an analysis for establishing the feasibility of distributed real-time systems. In such systems, among the most challenging requirements we find the so called *end-to-end* timing constraints. There is an end-to-end timing constraint for each couple of communicating tasks[1]: it is intended as the maximum time available for producing a message at the sender end, transmitting the message over the network, and processing it at the receiving end.

Given the specification of a system corresponding to the model described in the previous section, we will describe an analysis able to establish whether the *end-to-end* timing constraints are going to be met. The analysis is thus intended to provide a powerful tool to real-time designers for establishing the feasibility of a system before the system is actually built, and to give feedbacks about its temporal characteristics during the development phases.

As already addressed in [24], the time needed for a whole end-to-end computation is composed of five major components, depicted in Figure 2:

- *Generation delay* - the time needed by the sender task to generate and queue the message.
- *Queuing delay* - the time needed by the message to gain access to the network.
- *Transmission delay* - the time needed for the transmission of the message.
- *Delivery delay* - the time needed by the destination processor to deliver the message to the destination task.
- *Processing delay* - the time needed by the destination task to process the message, that is, to complete its execution.

Tindell and Clark [23] have described a very interesting approach for the analysis of end-to-end computations in real-time distributed systems where host processors schedule tasks and messages according to fixed priorities. They have termed this approach *holistic*, since it addresses the problem of analysing a system as a whole. Owing to its proven effectiveness, here we take the same holistic approach and we apply it to systems where earliest deadline dispatchers are used locally, which should let achieve a higher system-wide resource utilization.

In our opinion, the major original contribution of the holistic approach is to have elegantly overcome the difficulty of a global distributed analysis by means of a very simple concept: *attribute inheritance*. The message sent by a task inherits from it two of its temporal attributes, namely the period and the release jitter. If each instance of the task communicates, the message inherits a period equal to that of the task. Furthermore, if the message can be queued at any time by the sender task, the difference from its earliest and

[1] More generally, we can have a timing constraint for a whole "chain" of communicating tasks.

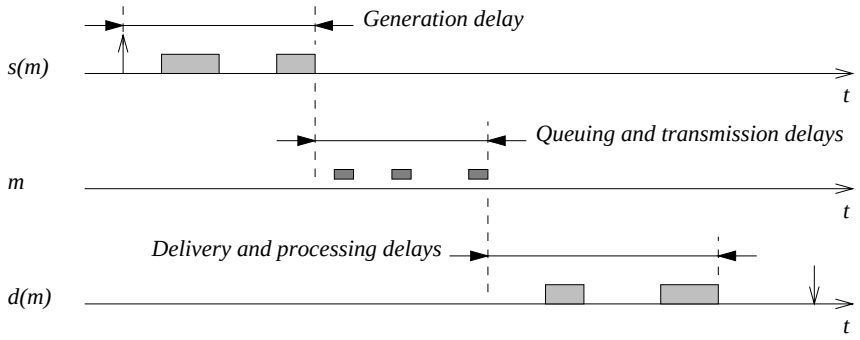

Figure 2: End-to-end computation delay components.

latest releases is bounded by the sender worst-case response time. This is the release jitter inherited by the message. Similarly, the destination task inherits its period and release jitter from the message it receives.

In this way, the overall analysis can be decoupled in several simpler analysis of smaller subsystems, namely the host processors and the network. Given their attributes, these subsystems can be analysed by means of exact procedures that let us find the worst-case response times of tasks and messages. By suitably combining these values, we can find tight bounds for end-to-end computations and we can then compare them with the relative constraints, in order to establish the feasibility of the whole system.

As it may have been noticed, however, the analysis of the different subsystems are mutually dependent: the release jitter of a message depends on the response time of its sender, the release jitter of the destination task depends on the response time of the message. Fortunately, as it will be shown in the following sections, this dependency is expressed in terms of non-decreasing functions. This lets us develop an analysis based on successive recursive steps, as illustrated in the following equations:

$$\begin{cases} R_1^{(m+1)} &= \mathcal{R}_{edf}\left(J_1^{(m)}\right) \\ \vdots & \\ R_n^{(m+1)} &= \mathcal{R}_{edf}\left(J_n^{(m)}\right) \\ R_{net}^{(m+1)} &= \mathcal{R}_{net}\left(J_{net}^{(m)}\right) \end{cases} \qquad \begin{cases} J_1^{(m+1)} &= \mathcal{P}_1\left(R_{net}^{(m+1)}\right) \\ \vdots & \\ J_n^{(m+1)} &= \mathcal{P}_n\left(R_{net}^{(m+1)}\right) \\ J_{net}^{(m+1)} &= \mathcal{P}_{net}\left(R_1^{(m+1)}, \ldots, R_n^{(m+1)}\right) \end{cases}$$

At each step, the worst-case response times of tasks and messages are computed for the n host processors and for the network, assuming the release jitters, and the other known attributes of course, computed in the previous step. The values of the release jitters are then updated before the next step.

In this way, the computation is halted either when the equations stabilize, that is, when the values of worst-case response times and release jitters do not further change, or at least one subsystem is found unschedulable.

Note that the computation of worst-case response times in the several subsystem is an essential tool for the holistic approach. A procedure for such a computation in uniprocessor systems with earliest deadline scheduling has been described in [20], and will be recalled in the following section. Later, we will show how to compute the worst-case queuing and transmission delays of messages scheduled according to the timed token protocol previously described. Finally, the two techniques will be integrated in order to find actual end-to-end computation delays.

4 Uniprocessor EDF Scheduling

In this section we will briefly recall the procedures for analysing the feasibility and computing the worst-case response times of a set of tasks scheduled in a uniprocessor system according to the Earliest Deadline First algorithm. The interested reader can find a more complete description in [20].

4.1 Feasibility Analysis

The rationale behind our procedure is described in the following theorem, which is the generalization of an original result described by Liu and Layland [14].

Theorem 4.1 *When the deadline driven scheduling algorithm is used to schedule a set of tasks on a processor, if there is an overflow for a certain arrival pattern, then there is an overflow without idle time prior to it in the pattern in which all task instances are released as soon as possible (according to their attributes).*

According to the theorem we simply need to study the schedule of the most demanding arrival pattern in the first *busy period*, *i.e.* in the first interval from time $t = 0$ up to the first processor idle time. The concept of busy period, already known in the literature [13], is also very useful for the computation of the worst-case response times, as we will shortly see.

The length L of the busy period can be computed by means of a simple iterative formula:

$$\begin{cases} L^{(0)} & = & \sum_{i=1}^{n} C_i, \\ L^{(m+1)} & = & W(L^{(m)}), \end{cases} \tag{1}$$

where $W(t)$ is the cumulative workload at time t, *i.e.* the sum of the computation times of the task instances arrived before time t:

$$W(t) = \sum_{i=1}^{n} \left\lceil \frac{t + J_i}{T_i} \right\rceil C_i.$$

The computation in Equation (1) is stopped when two consecutive values are found equal, that is, $L^{(m+1)} = L^{(m)}$. L is then set to $L^{(m)}$. It can be easily proven that the sequence $L^{(m)}$ converges to L in a finite number of steps as long as the overall processor utilization of the task set is less then or equal to 1, that is, if

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1.$$

The feasibility of the task set can then be established by checking whether there are missed deadlines in the busy period. The actual deadline d of a task's instance scheduled within L must be preceded by the execution of all task instances with deadline before or at d. That is, a sufficient condition for the feasibility of the task set is that for all actual deadlines d within L of a task's instance

$$d \geq \sum_{D_i \leq d+J_i} \left(1 + \left\lfloor \frac{d + J_i - D_i}{T_i} \right\rfloor\right) C_i \; + \; B_{k(d)}, \tag{2}$$

where [2]

$$k(d) = \max\{k : \; D_k - J_k \leq d\}.$$

4.2 Worst-Case Response Time Computation

The worst-case response time r_i of a task i is the maximum time between an i's instance arrival and its completion. As already stated, the computation of r_i is a fundamental tool for the holistic analysis of a distributed system: according to our assumptions, the response time of a task is the maximum jitter of the messages it sends over the network.

Finding r_i is not a trivial task when EDF scheduling is assumed. In fact, contrary to our intuition, the worst-case response time of a task is not always found in the first busy period. This is true for fixed priority scheduling, where the notion of *critical instant* has been known since the presentation of Liu and Layland's work [14], but not for EDF scheduling, as pointed out in [20]. The equivalent of the critical instant for a deadline scheduled task set is not necessarily when all tasks all released synchronously, and in general it is also different for each task.

The concept of busy period, however, helps us solving the problem. The idea is that the completion time of a task's instance with deadline d, must be the end of a busy period in which all executed instances have deadlines less than or equal to d. If we are able to examine all such periods, by taking the maximum length we can find the worst-case response time of a task. The following lemma characterizes the interesting busy periods.

Lemma 4.1 *The worst-case response time of a task i is found in a busy period in which all other tasks are released synchronously at the beginning of the period and then at their maximum rate (see Figure 3b).*

[2] We assume the tasks are ordered by non decreasing values of $D_i - J_i$, that is, $i < j \;\Rightarrow\; D_i - J_i \leq D_j - J_j$.

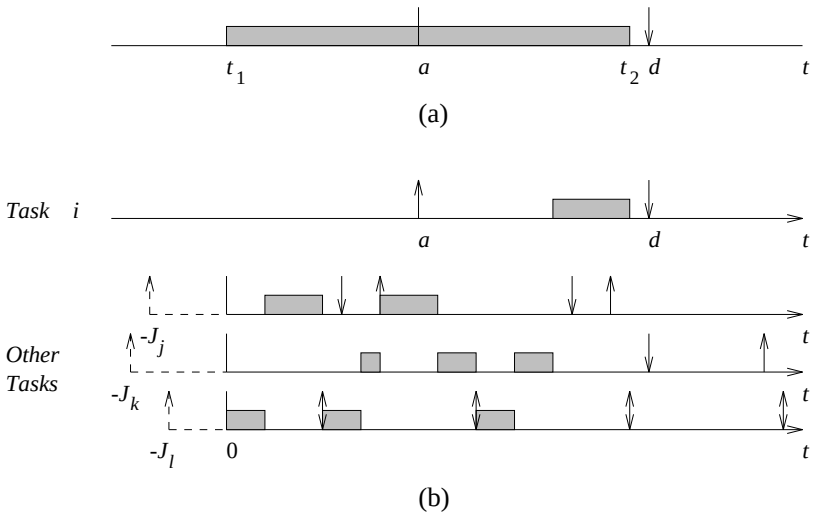

Figure 3: a) Busy period preceding an instance completion time. b) Arrival pattern possibly giving the worst-case response time for task i.

The lemma gives us the algorithm for computing the worst-case response time of a task i: we only need to compute the length of the busy periods of task instances with deadlines less than or equal to that of the i's instance considered, for a number of arrival patterns like the one shown in Figure 3b, where instance arrivals are represented by upward arrows and deadlines by downward arrows. All tasks but i are released synchronously and at their maximum rate from time $t = 0$. Our attention will be on the i's instance released at time $t = a$, $a \geq 0$, and possibly preceded by other instances of task i (these instances may contribute to increase the busy period length).

In particular, given $a \geq -J_i$, we consider an arrival pattern with all possible i's instances, so that there is one arrival at time $t = a$. In order to include the most possible instances, we force the first one to experience a release jitter J_i and we require its release time to be greater than or equal to 0 (its ideal arrival time may be before time 0). In this way, the first i's instance has release time

$$s_i(a) = a + J_i - \left\lfloor \frac{a + J_i}{T_i} \right\rfloor T_i.$$

Since we are only interested in those busy periods that includes all i's instances, from that released at time $t = s_i(a)$ to that with arrival time $t = a$, we can immediately take into account in the computation of the busy period length the overall workload of all such instances, which is

$$\left(1 + \left\lfloor \frac{a + J_i}{T_i} \right\rfloor\right) C_i.$$

If all other tasks are initially released at time $t = 0$, at time t, $\left\lceil \frac{t+J_j}{T_j} \right\rceil$ instances of j will have been released, for each $j \neq i$ (recall that all tasks ideally experience their maximum jitter at their first arrival). However, at most only $1 + \left\lfloor \frac{a+D_i+J_j-D_j}{T_j} \right\rfloor$ of them can have a deadline less than or equal[3] to $d = a + D_i$. That is, the higher priority workload, relative to deadline d, arrived up to time t is

$$W_i(a,t) = \sum_{\substack{j \neq i \\ D_j \leq a + D_i + J_j}} \min \left\{ \left\lceil \frac{t+J_j}{T_j} \right\rceil , 1 + \left\lfloor \frac{a+D_i+J_j-D_j}{T_j} \right\rfloor \right\} C_j.$$

The length $L_i(a)$ of the resulting busy period relative to the deadline d can then be computed with the following iterative formula:

$$\begin{cases} L_i^{(0)}(a) & = & \sum_{\substack{j \neq i \\ D_j \leq a + D_i + J_j}} C_j, \\ L_i^{(m+1)}(a) & = & W_i\left(a, L_i^{(m)}(a)\right) + \left(1 + \left\lfloor \frac{a+J_i}{T_i} \right\rfloor\right) C_i + B_{k(a+D_i)}. \end{cases} \quad (3)$$

As for Equation (1), the convergence of Equation (3) in a finite number of steps is ensured by the condition

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1.$$

Once we have determined $L_i(a)$, the worst-case response time relative to a is

$$r_i(a) = \max\left\{J_i + C_i + B_i, L_i(a) - a\right\}.$$

Finally, according to Lemma 4.1, the worst-case response time of task i is

$$r_i = \max_{a \geq -J_i} \left\{r_i(a)\right\}. \quad (4)$$

Fortunately, we do not need to evaluate $r_i(a)$ for all $a \geq -J_i$ in Equation (4). In fact, it is known that L, the length of the first busy period, is the maximum of all busy period lengths. Hence, the significant values of a are in the interval $[-J_i, L - J_i - C_i - B_i[$. Furthermore, it is not difficult to see that the local maxima of $L_i(a)$ are found for those values of a such that in the arrival pattern there is at least one instance of a task different from i with deadline equal to d, or all tasks are synchronous, *i.e.* $s_i(a) = 0$. These considerations let us speed significantly up the evaluation of Equation (4).

[3]We do not assume any particular policy for breaking deadline ties. Hence, in the worst-case instances sharing the same deadline should be considered as having higher priorities.

5 Communications Analysis

As already mentioned in Section 3, the *attribute inheritance* is a fundamental aspect of the holistic analysis. Accordingly, a message m inherits from its sender task $s(m)$ period and release jitter: $T_m = n_m T_{s(m)}$, if m is sent once every n_m periods of $s(m)$, and $J_m = r_{s(m)}$, the worst case response time of $s(m)$ [4]. This technique contributes to simplify the analysis, since it lets isolate the network subsystem, as far as the computation of message worst-case communication delays is concerned. By communication delay of a message we intend the sum of its queuing and transmission delays.

A procedure for this computation can be determined following an approach similar to that described in the previous section for the application tasks. In fact, we have assumed that messages are locally scheduled by the communication adapters according to the EDF algorithm. Thus the argument of Lemma 4.1 can be also extended to communication scheduling. The reason is that messages are locally scheduled in the same way as tasks on a uniprocessor. The main difference is the network access, which is governed by the Timed Token protocol. However, we will briefly see that this can be taken into account by locally considering the time not available to the transmission of synchronous messages as higher priority interference, which in turn can be precisely bounded.

The worst-case communication delay of a messame m is thus

$$r_m = \max_{a \geq -J_m} \{r_m(a)\}, \tag{5}$$

the maximum among all scenarios like that shown in Figure 4, in which one occurrence of message m arrives at time $t = a$ and all other messages are released, locally and remotely, synchronously and at their maximum rate. Later in this section we will discuss how to bound the values of a for which $r_m(a)$ has to be evaluated in Equation (5).

Since the transmission of packets is non-preemptable, in order to effectively evaluate the communication delay of message m, $r_m(a)$, we follow the approach described in [24]: we separate the queuing and the transmission delays of m by exactly bounding the communication delay of the first $k-1$ packets and the transmission delay of the kth packet. The idea is to exactly bound the worst-case time needed by the kth packet to gain access to the network and then to include the time needed for its transmission to the destination end. In practice, the communication delay of the first k packets of m is thus

$$r_{m,k}(a) = \max\{J_m + B_m + k\rho + \mathcal{P}, L_{m,k}(a) + \rho + \mathcal{P} - a\},$$

where $L_{m,k}(a)$ is the length of the higher priority network busy period which precedes the transmission of the kth packet of m, that is, the maximum time needed by the kth packet to reach the network, while $\rho + \mathcal{P}$ is the time for its transmission and for its propagation

[4] More generally, the release jitter of the message could be smaller if we are able to find a minimum relative queuing time greater than zero, and a maximum relative queuing time less than $r_{s(m)}$. The release jitter of message m is the difference between the two values.

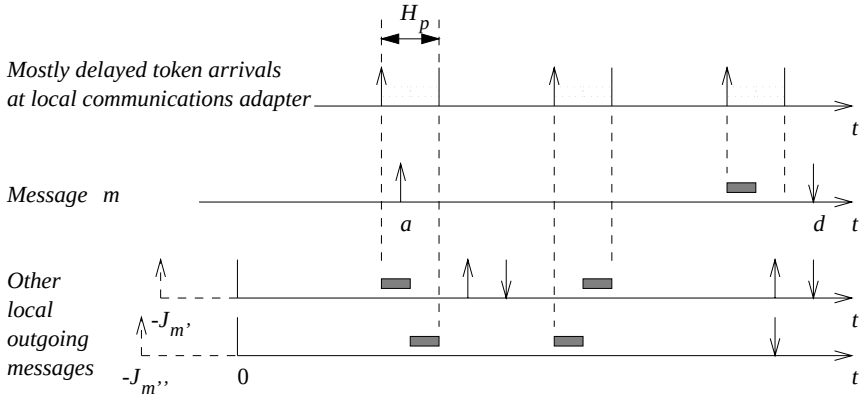

Figure 4: Local network scenario for the evaluation of message worst-case communication delays.

through the network to its destination. B_m is the blocking time of m: due to the non-preemptive character of packets transmission, m may be delayed by the transmission of a lower priority packet. Hence, $B_m = \rho$.

The maximum among the two quantities is necessary because $J_m + B_m + k\rho + \mathcal{P}$ is an obvious lower bound to the value of $r_{m,k}(a)$. The worst-case communication delay of the whole message m can be easily computed by substituting $k = C_m$, that is, $r_m(a) = r_{m,C_m}(a)$.

The higher priority network busy period preceding the transmission of m's kth packet can be divided in three components:

- the local higher priority workload of messages which have deadlines before or at $a+D_m$;
- the previous packets of message m;
- the interference of other hosts and of the possible local asynchronous traffic on the synchronous network access.

The first two components do not depend on the particular version of the protocol we assume. Thus we do not need to make a distinction.

The higher priority workload locally preceding the transmission of m's kth packet can be exactly taken into account by defining the function $\overline{HW}_m(a,t)$. In the interval $[0,t]$ a message m' is released $1 + \left\lfloor \frac{t+J_{m'}}{T_{m'}} \right\rfloor$ times, but at most $1 + \left\lfloor \frac{a+D_m+J_{m'}-D_{m'}}{T_{m'}} \right\rfloor$ occurrences have deadlines less than or equal to $a + D_m$, thus

$$\overline{HW}_m(a,t) = \rho \sum_{\substack{m' \neq m \\ m' \in Out(p) \\ D_{m'} \leq a + D_m + J_{m'}}} \min\left\{1 + \left\lfloor \frac{t + J_{m'}}{T_{m'}} \right\rfloor, 1 + \left\lfloor \frac{a + D_m + J_{m'} - D_{m'}}{T_{m'}} \right\rfloor\right\} C_{m'},$$

where $Out(p)$ is the set of outgoing messages of host processor p, with $m \in Out(p)$.

Note that in the equation that defines $r_{m,k}(a)$, owing to the non-preemptability of packet transmissions, we have separated the queuing time of the kth packet, $L_{m,k}(a)$, from its transmission dealy to the destination end, $\rho + \mathcal{P}$. However, this implies that the definition of $\overline{HW}_m(a,t)$ has to include also higher priority messages possibly released at time t, since they will precede the transmission of the packet we are looking at [5].

The second component of the higher priority network busy period can be easily accounted for. In the scenario we are examining, the occurrence of message m arrived at time $t = a$ is preceded by $\left\lfloor \frac{a+J_m}{T_m} \right\rfloor$ other occurrences, hence the kth packet is preceded by

$$\left\lfloor \frac{a+J_m}{T_m} \right\rfloor C_m + k - 1$$

other packets of the same message. Since we are only interested in busy periods whose lengths are greater than a, the transmission time of all these packets may be included as a whole in the definition of $L_{m,k}(a)$.

The third and final component is essentially due to token visits delayed as much as possible by the network accesses of other hosts. The maximum delay of any visit can be exactly computed by carefully analysing the protocol. In the following subsections we propose two different analysis for the full and the restricted versions of the protocol, respectively.

5.1 Full Timed Token Protocol

When asynchronous service is allowed at any host, subsequent token visits at node p may be delayed by synchronous and asynchronous packets transmitted by other hosts, as well as by local asynchronous packets. Zhang and Burns [25] have derived an exact upper bound on the time between any v consecutive token arrivals at host processor p:

$$t_{l+v-1,p} - t_{l,p} \leq (v-1)TTRT + \sum_{q \neq p} H_q + \tau - \left\lfloor \frac{v-1}{n+1} \right\rfloor \left(TTRT - \sum_{q=1}^{n} H_q - \tau \right),$$

where $t_{l,p}$ is the time the token makes its lth arrival at host p, and τ is the portion of $TTRT$ unavailable for transmitting messages. Note that the "equal" sign holds in the worst-case. The visits of the token at host p, useful for synchronous transmissions, are then mostly delayed when:

- the first visit occurs at time $t = 0^-$, that is, just before any synchronous packet has been released, thus making useless this first visit, and
- the asynchronous capacity is fully utilized at any host, whenever available.

[5] Note that in case of preemptive scheduling the transmission time of the kth packet would be included in $L_{m,k}(a)$, but the definition of $\overline{HW}_m(a,t)$ would include only those higher priority messages released before t.

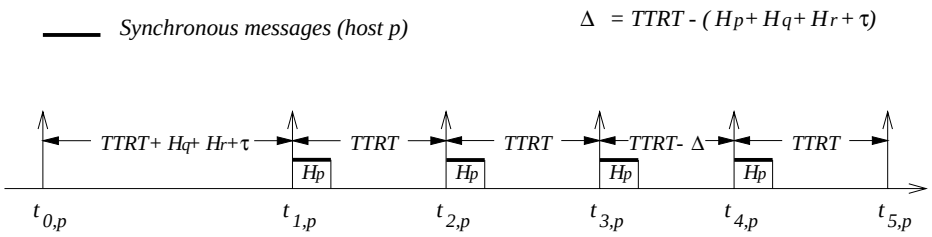

Figure 5: Mostly delayed token visits at host processor p when full Timed Token protocol is assumed.

See Figure 5 for a graphical representation, where the visits useful for synchronous message transmissions at host p are $t_{1,p}, t_{2,p}, \ldots$ Accordingly, if v is such that

$$t_{v-1,p} + H_p \;\leq\; t \;<\; t_{v,p} + H_p,$$

the interference of the other processors and the local asynchronous traffic in the interval $[0,t]$ can then be defined as

$$\bar{I}_p(t) = t_{v,p} - (v-1)H_p,$$

where

$$t_{v,p} = v\,TTRT \;+ \sum_{q \neq p} H_q + \tau - \left\lfloor \frac{v}{n+1} \right\rfloor \left(TTRT \;- \sum_{q=1}^{n} H_q - \tau \right).$$

The length of the higher priority network busy period can be finally computed by solving the equation

$$L_{m,k}(a) = \overline{HW}_m\left(a, L_{m,k}(a)\right) \;+\; \left(\left\lfloor \frac{a + J_m}{T_m} \right\rfloor C_m + k - 1 \right) \rho \;+\; \bar{I}_p\left(L_{m,k}(a)\right). \qquad (6)$$

Being $\bar{I}_p(t)$ and $\overline{HW}_m(a,t)$ both monotonically non-decreasing step functions, Equation (6) can be practically solved as usual by means of an iterative fixed point computation:

$$\begin{cases} L^{(0)}_{m,k}(a) &= 0, \\ L^{(i+1)}_{m,k}(a) &= \overline{HW}_m\left(a, L^{(i)}_{m,k}(a)\right) \;+\; \left(\left\lfloor \frac{a+J_m}{T_m} \right\rfloor C_m + k - 1 \right) \rho \;+\; \bar{I}_p\left(L^{(i)}_{m,k}(a)\right). \end{cases}$$

The computation is halted when two consecutive values are found equal.

5.2 Restricted Timed Token Protocol

When asynchronous service is not allowed, the analysis becomes in some way simpler. Without loss of generality, we can now assume that

$$TTRT \;=\; \sum_{p=1}^{n} H_p \;+\; \tau.$$

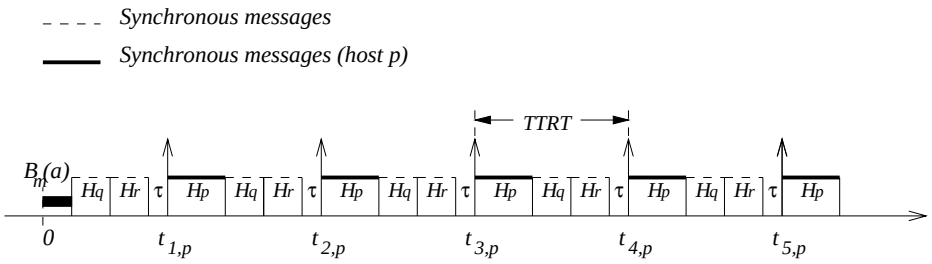

Figure 6: Mostly delayed token visits at host processor p when restricted Timed Token protocol is assumed.

The worst-case condition which leads to mostly delayed token visits is depicted in Figure 6. Note that the first visit of the token to host p may be partially "wasted" for the transmission of a lower priority packet, which is not preemptable. This potential blocking time $B_m(a)$ is thus at most the transmission time ρ of a single packet. However, it must be accounted for only if in host processor p there are messages that when queued at time $t = 0^-$ they cannot have an actual deadline before or at $a + D_m$ (*i.e.* messages not included in $\overline{HW}_m(a,t)$):

$$B_m(a) = \begin{cases} \rho & \text{if } \exists m' \in Out(p): \ D_{m'} > a + D_m + J_{m'} \\ 0 & \text{otherwise.} \end{cases}$$

Note that in the busy period a blocking relative to $a + D_m$ can only occur at the beginning.

Similarly to what seen in the previous subsection, the worst-case interference of the other host processors on the network accesses of host processor p in an interval of time t can be exactly computed by carefully defining the function $\bar{I}_p(t)$. Each host q has a *synchronous bandwidth* H_q. Consequently, by using a similar argument as for the local higher priority workload, in the interval $[0,t]$ the interference of q is upper bounded by

$$\left(1 + \left\lfloor \frac{t}{TTRT} \right\rfloor\right) H_q.$$

However, this upper bound may be pessimistic because it assumes that the communications adapter at host processor q is always busy transmitting messages for a period of time H_q whenever it receives the token. This may not be true if there are not enough outgoing messages queued by q. Since we know exactly the queuing law of these messages, we are able to determine a second bound on the interference of q in the interval $[0,t]$, by computing the message workload at its communications adapter in the same interval:

$$\overline{W}_q(t) = \rho \sum_{m' \in Out(q)} \left(1 + \left\lfloor \frac{t + J_{m'}}{T_{m'}} \right\rfloor\right) C_{m'}.$$

The actual bound is the minimum between the two values. The overall interference on host processor p is then

$$\bar{I}_p(t) = \left(1 + \left\lfloor \frac{t}{TTRT} \right\rfloor\right) \tau \; + \; \sum_{q \neq p} \min \left\{ \left(1 + \left\lfloor \frac{t}{TTRT} \right\rfloor\right) H_q, \overline{W}_q(t) \right\}.$$

Hence, in this case the equation for the computation of $L_{m,k}(a)$ is

$$L_{m,k}(a) = \overline{HW}_m \left(a, L_{m,k}(a)\right) + \left(\left\lfloor \frac{a + J_m}{T_m} \right\rfloor C_m + k - 1 \right) \rho + B_m(a) + \bar{I}_p \left(L_{m,k}(a)\right). \quad (7)$$

The solution of the equation can be found as previously by a fixed point computation.

5.3 Search Interval

The last issue to be addressed in this description is how to identify the significant values of the variable a in Equation (5). The argument we are going to use is the same as for the evaluation of task worst-case response times. Since in the computation of $r_{m,k}(a)$ we evaluate the length of a busy period, $L_{m,k}(a)$, we obtain an upper bound on the significant values of a by founding the maximum length of any possible adapter busy period, L_p.

The length L_p can be determined by taking into account the local message workload and the remote interference, and by using the well known fixed point computation:

$$\begin{cases} L_p^{(0)} & = & \rho \sum_{m \in Out(p)} C_m, \\ L_p^{(m+1)} & = & W_p\left(L_p^{(m)}\right) \; + \; I_p\left(L_p^{(m)}\right), \end{cases}$$

where $W_p(t)$ and $I_p(t)$, contrary to the corresponding $\overline{W}_p(t)$ and $\bar{I}_p(t)$, respectively, are defined on the interval $[0, t[$, that is, they only include instances arrived before t:

$$W_p(t) = \rho \sum_{m \in Out(p)} \left\lceil \frac{t + J_m}{T_m} \right\rceil C_m$$

and

$$I_p(t) = t_{v,p} - (v-1)H_p,$$

where v is such that

$$t_{v-1,p} + H_p \; < \; t \; \leq \; t_{v,p} + H_p,$$

if the full version of the Timed Token protocol is assumed,

$$I_p(t) = \left\lceil \frac{t}{TTRT} \right\rceil \tau \; + \; \sum_{q \neq p} \min \left\{ \left\lceil \frac{t}{TTRT} \right\rceil H_q, W_q(t) \right\},$$

if, vice versa, the restricted version is assumed.

The significant values of a are then found in the interval $[-J_m, L_p - J_m - B_m - \rho C_m[$. In order to further reduce the number of evaluations of $r_m(a)$, we can still observe that the local maxima of the higher priority busy period length $L_{m,k}(a)$ are found for those values of a for which there is at least another local message with one occurrence having the deadline at time $a+D_m$, or for which also message m has an occurrence queued, *i.e.* released, at time $t = 0$, that is, all messages are first released synchronously. Hence, the significant values of a are those in the interval $[-J_m, L_p - J_m - B_m - \rho C_m[$ for which $\exists m'$, $\exists k$ such that $a = -J_{m'} + kT_{m'} + D_{m'} - D_m$. These are in fact the only points at which the right sides of Equations (6) and (7) have discontinuities in a.

6 End-To-End Response Times

In the previous sections we have seen how the generation, queuing and transmission delays of an end-to-end computation can be tightly bounded. In particular, the generation delay is bounded by the worst-case response time of the sender task, while the queuing delay and the transmission delay are included in the worst-case message communication delay. The final step, which we are going to describe in this section, is to bound the delivery delay and the processing delay, that is, the time to deliver the message to the destination task in the destination processor and the time to complete its execution.

According to Tindell *et al.* [24], the delivery delay can be implicitly included in our analysis by accurately bounding the overhead needed to handle the interrupts raised by packet arrivals at a destination processor. The packet interrupt handler is in charge of message reconstruction at the receiving end, and message delivery, that is, release of the actual destination task, after the arrival of the last packet. By including these overheads in the computation of the worst-case response time of the destination task, we will bound at the same time the delivery and the processing delays of the end-to-end computation.

A first rough upper bound on the packet handling overhead can be found by observing that the packet transmission time, ρ, is the minimum time between two consecutive packet arrivals at any host processor. Hence in any interval of time t we have the following bound:

$$\left\lceil \frac{t}{\rho} \right\rceil C_{packet},$$

where C_{packet} is the worst-case execution time of the packet handler.

A second bound can be obtained by describing the packet handling relative to each message m as a *sporadically periodic task* [3], a model particularly suited for the analysis of "bursty" tasks. According to this model, the interrupts handling m's packets are described by means of four parameters:

$n_{h(m)}$ The number of handler invocations in a "burst." In this case $n_{h(m)} = C_m$, the number of packets message m is composed of.

$t_{h(m)}$ The minimum time between arrivals within a burst. We have $t_{h(m)} = \rho$.

$T_{h(m)}$ The periodicity of the burst. It is equal to the period of message m, that is, $T_{h(m)} = T_m$.

$J_{h(m)}$ The release jitter of packet arrivals. It is the difference between the earliest and latest arrival of the whole message m, that is, $J_{h(m)} = r_m - (\rho C_m + \mathcal{P})$.

Using this description, Tindell *et al.* [24] have derived the following bound on the number of packets that can arrive at a host processor p in an interval of time t:

$$P_p(t) = \sum_{m \in In(p)} \left\lfloor \frac{t + J_{h(m)}}{T_{h(m)}} \right\rfloor n_{h(m)} + \min \left\{ n_{h(m)}, \left\lceil \frac{t + J_{h(m)} - \left\lfloor \frac{t+J_{h(m)}}{T_{h(m)}} \right\rfloor T_{h(m)}}{t_{h(m)}} \right\rceil \right\},$$

where $In(p)$ is the set of incoming messages in host processor p. The packet handling overhead at p can finally be bounded by the function

$$OV_p(t) = \min \left\{ \left\lceil \frac{t}{\rho} \right\rceil, P_p(t) \right\} C_{packet}. \tag{8}$$

As suggested by Tindell *et al.*, the packet handling overhead would be much less if the communications adapter had some processing capabilities and would be able to distinguish the last packet of a message among the others. In this way the host processor could be interrupted only once, after the arrival of the last packet, thus reducing significantly the delivery overhead.

The overhead described by Equation (8) can be included in the host processor analysis of Section 4, both for the feasibility evaluation and the worst-case response time computation aspects, as illustrated in [20], where the overhead of a tick driven scheduler is similarly taken into account. The approach is based on the simple idea that the overhead is just to be considered as higher priority load for any application task. Having tightly bounded such load, it is sufficient to accurately include it in all equations where we take into account all tasks having a higher priority, owing either to a shorter deadline or to a "different class" as in this case, in order to check a deadline or to compute a busy period length. We will not further describe this aspect, leaving the interested reader to the cited reference.

Our final step is at this point to bound the processing delay of the end-to-end computation by evaluating the worst-case response time of the destination task $d(m)$. This can be done as described in Section 4, with the extension just mentioned in the previous paragraph. We only have to describe the attributes inherited by $d(m)$. This is quite straightforward: the period $T_{d(m)}$ is equal to the period T_m of message m, while the release jitter is equal to the difference between the latest and the earliest arrival times of m, that is, $J_{d(m)} = r_m - (\rho C_m + \mathcal{P})$.

The description of our procedure is now complete. Note that there is a strong dependency between host processor and network scheduling. The attribute inheritance is such that the message communication delays strongly depend on the sender task response times and, vice versa, the worst-case response times of destination tasks strongly depend on message communication times. This dependency, already described in Section 3, is the base of the

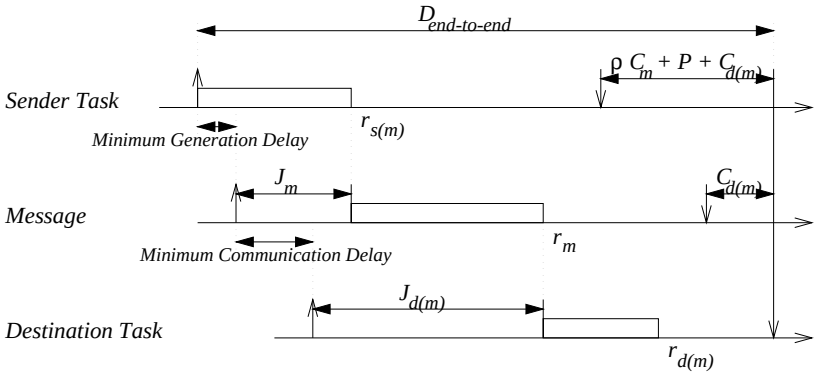

Figure 7: Components of the end-to-end computation delay.

holistic analysis. In order to coherently analyse the system as a whole we must proceed in iterative steps. At each step we examine all subsystems using the methodologies described in Section 4 for host processors and in Section 5 for the network, and the attributes computed at the previous step. The computation is halted either when all values stabilize or when a subsystem is found unschedulable.

Once the holistic procedure is halted, the last step is to check the worst-case end-to-end computation delays against their requirements. In particular, the five components of an end-to-end computation delay can be finally summed up as in the following formula:

$$r_{end\text{-}to\text{-}end} = r_{s(m)} + (r_m - J_m) + \left(r_{d(m)} - J_{d(m)}\right) . \tag{9}$$

The formula is also graphically illustrated in Figure 7. The worst-case response time of the sender task, $r_{s(m)}$, is the generation delay. $(r_m - J_m)$ includes the queuing and the transmission delays. The delivery delay (implicitly) and the processing delays are included in the last term $\left(r_{d(m)} - J_{d(m)}\right)$.

One last issue remains to be addressed. So far we have assumed that the distributed feasibility problem is specified in terms of maximum end-to-end computation delays. We have also assumed that the subsystems, host processors and local communications adapters, are scheduled according to the Earliest Deadline First algorithm. Thus the question is how to assign deadlines to intermediate steps of each end-to-end computation (sender task, message and destination task) in order to always achieve the maximum schedulability. In other words, we want to find a deadline assignment able to guarantee the feasibility of the system, whenever this is possible with EDF scheduling. Note that this is no longer a simple analysis of the system, but a helpful task aimed at a correct and effective design of the system.

At present we still do not have an answer to this problem. We believe, however, that a sensible deadline assignment can be easily found by looking at Figure 7. If $D_{end\text{-}to\text{-}end}$ is the relative deadline of the whole end-to-end computation, the destination task $d(m)$ is assigned

the relative deadline

$$D_{d(m)} = \min\left\{D_{d(m)}, D_{end\text{-}to\text{-}end} - mgd_m - mcd_m\right\},$$

where mgd_m is the minimum generation delay and mcd_m is the minimum communication delay of message m [6]. Note that the minimum is necessary to preserve a possible stronger constraint specified by the system designer.

Similarly, message m is assigned the relative deadline

$$D_m = \min\left\{D_m, D_{d(m)} - C_{d(m)} + mcd_m\right\}.$$

Finally, for the source task $s(m)$ we have

$$D_{s(m)} = \min\left\{D_{s(m)}, D_m - mcd_m + mgd_m\right\}.$$

Not only simple and intuitive, this assignment has also been found effective for the case study described in the following section. Although we believe it could be a reasonable choice for practical systems, we do not claim it is optimal. In fact, some preliminary experiments have shown that the assignment can indeed be improved. The idea is to shorten the deadlines of those tasks or messages for which we need shorter response times. How to do this consistently with the system requirements and without jeopardizing the other system components is not a trivial task. Being the subject of current research, we will not further address the problem in this paper and we will assume in the following the deadline assignment described above.

7 Case Study

In this section we describe a case study application, a hypothetical aircraft control system, that we have analysed by means of a simple tool in which we have implemented the theory presented in the paper. The example is taken from [23]. The only variations are the local task and message scheduling algorithms, which in our case are both Earliest Deadline First (Tindell and Clark assumed fixed priorities) and the network protocol, which we assume to be a restricted version of the Timed Token MAC protocol (Tindell and Clark assumed a TDMA). All the rest is unchanged, including the parameters of the network.

There are three hypothetical host processors in the example system. For all of them we have assumed the presence of a tick driven scheduler with a clock tick period of $1000\mu s$, computation time $66\mu s$, and costs for the task moves between the scheduler queues C_{QL} equal to $74\mu s$ and C_{QS} equal to $40\mu s$ (see [23] or [20] for an explanation of these terms).

For the communications we have assumed, as Tindell and Clark did, a packet size of 1024 bytes and a packet transmission time ρ of $800\mu s$. The synchronous bandwidths of host processors 1, 2, and 3 are equal to $800\mu s$ (1 packet), $800\mu s$ (1 packet), and $2400\mu s$ (3 packets) respectively. The ring latency and the protocol overheads are assumed to give a

[6] In this paper we have assumed $mgd_m = 0$ and $mcd_m = \rho C_m + \mathcal{P}$.

value of $\tau = 240\mu s$, thus giving a target token rotation time [7] $TTRT = 4240\mu s$, which is equal to the TDMA cycle time assumed by Tindell and Clark.

The timing characteristics of tasks allocated to host processors 1, 2, and 3 are given in Table 1, 3, and 5 respectively. The semaphores and the relative local locking patterns for the three host processors are listed in Table 2, 4, and 6. Finally, the timing characteristics of the messages utilized in the application are given in Table 7, while the end-to-end computations are described in Table 8.

In the tables regarding tasks and messages we have reported the relative deadlines computed according to the deadline assignment described in the previous section, while the periods are derived according to the attribute inheritance approach from the values initially specified. The task blocking times have been computed as explained in [20]. Release jitters and worst-case response times are the results of the holistic analysis. In Table 8 we have also reported the worst-case response times of the end-to-end computations, computed according to Equation 9.

As can be easily verified, only one end-to-end deadline is potentially missed, and even in this case the estimated maximum lateness is of only $308\mu s$. If we compare the worst-case response times under EDF and fixed priorities scheduling, respectively, we do not find a big difference. However, we do consider the result a confirmation of the better processor and network utilization achievable with the EDF algorithm, owing to the following reasons:

- Given the non-preemptability of a packet transmission, the communication delay of the first k packets of a message m is computed as the sum of the queuing and the transmission delays of the kth packet. In order to bound the queuing delay, Tindell and Clark [23], as well as Tindell *et al.* [24], have computed the interference of local higher priority packets and the interference of other processors on the time interval $[0, t[$. We believe this is correct for a preemptive system, not for a non-preemptive one. Thus, as described in Section 5, we have computed the interferences on the time interval $[0, t]$, which we believe to be the correct approach.

- Tindell and Clark have not considered any blocking time for the messages. We do have considered a blocking time equal at most to a single packet transmission, as mentioned also by Tindell *et al.* in [24].

Owing to these arguments, which we believe should be used to correct the approach of Tindell and Clark, we have a potentially bigger interference. Being the results obtained comparable, we argue that this makes them even more considerable and a confirmation of the higher resource utilization achievable with the EDF algorithm.

In Table 9 we have finally reported the end-to-end response times computed when the full version of the Timed Token protocol is assumed. Owing to the generally higher interference experienced by any host processor in worst-case conditions when accessing the network,

[7] Recall that in the restricted version of the Timed Token protocol we have assumed $TTRT = \sum_p H_p + \tau$ and absence of asynchronous traffic. Consequently, the target token rotation time becomes the worst-case token rotation time.

Task name	D_i	T_i	C_i	B_i	J_i	r_i
task1	5000	200000	2277	321	0	5308
deliver_air_fuse_data	14199	40000	420	321	12098	14507
deliver_air_data	19199	20000	496	321	11610	17564
deliver_air_data_update	19199	160000	552	321	8168	15951
task3	12000	25000	1423	354	0	8752
task5	50000	50000	3096	354	0	24621
deliver_radar	98399	100000	3220	354	46347	73020
deliver_radar_update	87199	800000	3220	354	35147	61820
task7	59000	59000	7880	354	0	33621
task9	100000	50000	1996	343	24621	60359
client2	197598	200000	1120	343	105783	142707
client1	100000	200000	520	343	0	38530
task11	197598	2000000	954	343	103509	141453
task17	193499	1000000	1990	343	0	42780
task13	198545	200000	1124	252	0	45781
task15	200000	200000	3345	0	0	47236

Table 1: Timing parameters of tasks allocated to host processor 1.

Semaphore	Locked by	Time held
radar_data	deliver_radar	354
radar_data	deliver_radar_update	354
air_data	deliver_air_data	321
air_data	deliver_air_data_update	256
buffer_mngmt	task13	477
gyro_data	task9	221
radar_data	task9	354
air_data	task9	321
air_data	deliver_air_fuse_data	321
gyro_data	task15	252
radar_data	task3	108
msg_mngmt	task17	343
msg_mngmt	task13	343
msg_mngmt	task5	343
msg_mngmt	task3	343
msg_mngmt	client1	343

Table 2: List of semaphores and locking pattern for host processor 1.

all values have increased. However, only two end-to-end computations do not meet their maximum delay requirements. Furthermore, the maximum lateness foreseen by the analysis is smaller than a millisecond, which is anyway a remarkable result.

Task name	D_i	T_i	C_i	B_i	J_i	r_i
task2	5000	25000	689	0	0	1779
deliver_health	17599	100000	550	0	10010	12589
task4	13779	40000	996	343	0	4418
task6	50000	50000	4967	410	0	10652
task8	80000	80000	9125	410	0	20437
task10	115000	100000	5120	410	0	26716
deliver_actr	199199	200000	654	343	62380	93144
server	196478	200000	2342	756	59130	90423
task12	196644	200000	3145	350	57330	90589
task14	198399	175000	2325	350	31792	67508
task16	999199	1000000	1455	0	66620	103507

Table 3: Timing parameters of tasks allocated to host processor 2.

Semaphore	Locked by	Time held
health_data	server	350
buffer_mngmt	task12	477
actuator_ctrl	task10	410
actuator_ctrl	task6	410
health_data	task12	756
health_data	task16	350
actuator_ctrl	deliver_actr	410
msg_mngmt	task4	343
msg_mngmt	task12	343
msg_mngmt	server	343

Table 4: List of semaphores and locking pattern for host processor 2.

8 Conclusions

In this paper we have proposed a novel approach, termed holistic, for the analysis of deadline scheduled real-time distributed systems. According to our assumptions, application tasks are scheduled on host processors earliest deadline first, and accesses to the communication medium are arbitrated by using the Timed Token MAC protocol. A further original contribution derives from the assumption of earliest deadline queuing for outgoing packets at any host communications adapter.

The holistic analysis is based on an iterative process in which the computation of worst-case response times of tasks and messages is the essential tool utilized at each step. For application tasks the computation can be done as described in [20]. In this paper we have proposed a procedure for the computation of worst-case message communication delays, when either a full or a restricted version (*i.e.* only the synchronous service is allowed at any node) of the Timed Token MAC protocol is assumed for network accesses.

Task name	D_i	T_i	C_i	B_i	J_i	r_i
send_health	17049	100000	2322	343	0	3930
send_air	18647	20000	2245	343	0	5528
send_radar	83979	100000	12224	0	0	18267

Table 5: Timing parameters of tasks allocated to host processor 3.

Semaphore	Locked by	Time held
msg_mngmt	send_air	343
msg_mngmt	send_health	343
msg_mngmt	send_radar	343

Table 6: List of semaphores and locking pattern for host processor 3.

Message name	(cpu) Source task	(cpu) Destination task	n_m	C_m	D_m	T_m	J_m	r_m
air_data	(3) send_air	(1) deliver_air_data	1	1	19504	20000	5528	12411
air_data_update	(3) send_air	(1) deliver_air_data_update	8	1	19448	160000	5528	8969
health_data	(3) send_health	(2) deliver_health	1	3	19450	100000	3930	12411
radar_data	(3) send_radar	(1) deliver_radar	1	2	96780	100000	18267	47948
radar_data_update	(3) send_radar	(1) deliver_radar_update	8	16	96780	800000	18267	47948
message1	(1) task17	(2) task16	1	1	998545	1000000	42780	67421
message2	(1) task13	(2) deliver_actr	1	1	199346	200000	45781	63181
message3	(2) task4	(1) deliver_air_fuse_data	1	1	14580	40000	4418	12899
message4	(1) task5	(1) task9	1	1	98805	50000	24621	0
message5	(1) task17	(2) task12	2	2	195100	2000000	42780	58931
message6	(1) task3	(2) task14	7	2	197675	175000	8752	33393
message7	(2) task12	(1) task11	1	1	197445	2000000	90589	103310
toserver	(1) client1	(2) server	1	1	194937	200000	38530	58931
fromserver	(2) server	(1) client2	1	2	198079	200000	90423	107384

Table 7: Set of messages for our case study application.

The global deadline scheduling approach have been proven effective by means of an extended example, to which the holistic theory described in the paper has been applied. However, we believe improvements on the scheme are achievable by further investigations. The deadline assignment described in Section 6 may not be necessarily optimal. Late end-to-end computations may have their response times improved by shortening the deadlines of intermediate tasks and messages. This may give rise to a potentially exponential search space that could be explored by branch-and-bound techniques, as proposed in [1]. Furthermore, the deadline assignment is also linked to the task allocation strategy. Since the problem of tasks allocation is usually tackled by means of suitable heuristics, it would be interesting to design heuristics that take into account both aspects at the same time.

End-to-end computation components	Deadline	Response time
send_air → *air_data* → deliver_air_data	20000	18365
send_air → *air_data_update* → deliver_air_data_update	20000	16752
send_health → *health_data* → deliver_health	20000	14990
send_radar → *radar_data* → deliver_radar	100000	74621
send_radar → *radar_data_update* → deliver_radar_update	100000	74621
task17 → *message1* → task16	1000000	104308
task13 → *message2* → deliver_actr	200000	93945
task4 → *message3* → deliver_air_fuse_data	15000	15308
task5 → *message4* → task9	100000	60359
task17 → *message5* → task12 → *message7* → task11	200000	142855
task3 → *message6* → task14	200000	69109
client1 → *toserver* → server → *fromserver* → client2	200000	144109

Table 8: End-to-end computations and their timing parameters.

End-to-end computation components	Deadline	Response time
send_air → *air_data* → deliver_air_data	20000	20994
send_air → *air_data_update* → deliver_air_data_update	20000	19205
send_health → *health_data* → deliver_health	20000	17579
send_radar → *radar_data* → deliver_radar	100000	77638
send_radar → *radar_data_update* → deliver_radar_update	100000	77638
task17 → *message1* → task16	1000000	121118
task13 → *message2* → deliver_actr	200000	97990
task4 → *message3* → deliver_air_fuse_data	15000	15994
task5 → *message4* → task9	100000	60359
task17 → *message5* → task12 → *message7* → task11	200000	148061
task3 → *message6* → task14	200000	74149
client1 → *toserver* → server → *fromserver* → client2	200000	151115

Table 9: End-to-end computations parameters with the full version of the Timed Token protocol.

Another aspect which deserves further investigations is the *offset scheduling*: tasks are actually released at fixed offsets after their arrival. The technique has been proposed by Tindell and Clark [23] to allow single tasks to be destinations of multiple messages. We are not aware of works dealing with offset scheduling in the context of deadline scheduled systems.

Scheme improvements are also probably achieved by reducing the release jitter of tasks and messages, which should cause a shortening of worst-case response times. Tindell *et al.* [24] have proposed few methods to achieve this goal, among which the most effective one is based on the mentioned offset scheduling. The same techniques can also be applied to deadline scheduling. Note however, that a jitter reduction should be accompanied by a similar deadline reduction, in order to preserve the *preemption level*, corresponding to the difference between the two values [20], which in deadline scheduled systems plays a role very similar to that of priorities in fixed priority systems [4].

A final remark is about the synchronous bandwidth allocation for the Timed Token MAC protocol. As already mentioned in Section 2, several authors have described a number of allocation schemes able to guarantee the timely delivery of periodic messages when the global network utilization is lower than the scheme-dependent worst-case achievable utilization. Unfortunately, one of the assumption is that at each node there is a single synchronous stream, or multiple independent outgoing queues in case of multiple streams. We believe further investigations aimed at the minimization of maximum communications delays, when single deadline ordered outgoing queues are assumed, would be valuable.

References

[1] T.F. Abdelzaher and K.G. Shin, "Optimal Combined Task and Message Scheduling in Distributed Real-Time Systems," *Proc. of the IEEE Real-Time Systems Symposium*, 1995.

[2] G. Agrawal, B. Chen, W. Zhao, and S. Davari, "Guaranteeing Synchronous Message Deadlines with the Timed Token Medium Access Control Protocol," *IEEE Trans. on Computers* 43(3), March 1994.

[3] Audsley N., Burns A., Richardson M., Tindell K., and Wellings A.J., "Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling," *Software Engineering Journal*, September 1993.

[4] T.P. Baker, "Stack-Based Scheduling of Real-time Processes," *The Journal of Real-Time Systems* 3, pp. 67-99, 1991.

[5] R. Bettati and J.W.S. Liu, "End-to-End Scheduling to Meet Deadlines in Distributed Systems," *Proc. of the 12th Distributed Computing Systems Conference*, 1992.

[6] M. Chen and K. Lin, "Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time Systems," *The Journal of Real-Time Systems* 2, pp. 325-346, 1990.

[7] W.W. Chu, C. Sit, and K.K. Leung, "Task Response Time For Real-Time Distributed Systems With Resource Contentions," *IEEE Trans. on Software Engineering* 17(10), October 1991.

[8] M. Di Natale and J.A. Stankovic, "Dynamic End-to-end Guarantees in Distributed Real-Time Systems," *Proc. of the IEEE Real-Time Systems Symposium*, 1994.

[9] D. Ferrari, "A New Admission Control Method for Real-Time Communication in an Internetwork," in S. Son, Ed., *Advances in Real-Time Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1995.

[10] R.M. Grow, "A Timed Token Protocol for Local Area Networks," *Proc. Electro/82*, May 1982.

[11] R. Jain, *FDDI Handbook: High-Speed Networking Using Fiber and Other Media*, Addison-Wesley, Reading, Massachusetts, 1994.

[12] H. Kopetz *et al.*, "Distributed Fault-Tolerant Real-Time Systems: The MARS Approach," *IEEE Micro* 9(1), February 1989.

[13] Lehoczky J.P., "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines," *Proc. of the 11th IEEE Real-Time Systems Symposium*, December 1990.

[14] C.L. Liu and J.W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of ACM* 20(1), pp. 40-61, January 1973.

[15] N. Malcolm and W. Zhao, "Guaranteeing Synchronous Messages with Arbitrary Deadline Constraints in an FDDI Network," *Proc. of the IEEE Conf. on Local Computer Networks*, 1993.

[16] Y. Oh and S.H. Son, "Allocating Fixed-Priority Periodic Tasks on Multiprocessor Systems," *The Journal of Real-Time Systems* 9, 1995.

[17] D.T. Peng and K.G. Shin, "Static Allocation of Periodic Tasks with Precedence," in *Distributed Computing Systems*, IEEE, June 1989.

[18] K. Ramamritham, "Allocation and Scheduling of Precedence-Related Periodic Tasks," *IEEE Trans. on Parallel and Distributed Systems* 6(4), April 1995.

[19] L. Sha, R. Rajkumar, and J.P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Transactions on Computers* 39(9), September 1990.

[20] M. Spuri, "Analysis of Deadline Scheduled Real-Time Systems," Rapport de recherche 2772, INRIA Rocquencourt, Le Chesany Cedex, France, January 1996, submitted to *IEEE Trans. on Software Engineering.*

[21] J.A. Stankovic and K. Ramamritham, "The Spring Kernel: a new Paradigm for Real-Time Systems," *IEEE Software*, May 1991.

[22] J.A. Stankovic, M. Spuri, M. Di Natale, and G.C. Buttazzo, "Implications of Classical Scheduling Results for Real-Time Systems," *IEEE Computer*, June 1995.

[23] K. Tindell and J. Clark, "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems," *Microprocessors and Microprogramming* 40, 1994.

[24] K. Tindell, A. Burns, and A.J. Wellings, "Analysis of Hard Real-Time Communications," *The Journal of Real-Time Systems* 9, 1995.

[25] S. Zhang and A. Burns, "An Optimal Synchronous Bandwidth Allocation Scheme for Guaranteeing Synchronous Message Deadlines with the Timed-Token MAC Protocol," *IEEE/ACM Trans. on Networking* 3(6), December 1995.

[26] Q. Zheng and K.G. Shin, "Synchronous Bandwidth Allocation in FDDI Networks," *Proc. of ACM Multimedia 93*, 1993.

[27] Q. Zheng and K.G. Shin, "On the Ability of Establishing Real-Time Channels in Point-to-Point Packet-Switched Networks," *IEEE Trans. on Communications* 42(2/3/4), 1994.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399