Non Pre-emptive Scheduling of Messages on SMTV Token-Passing Networks

Eduardo Tovar

Department of Computer Engineering,
ISEP, Polytechnic Institute of Porto, Portugal
E-mail: emt@dei.isep.ipp.pt

Francisco Vasques

Department of Mechanical Engineering,
FEUP, University of Porto, Portugal
E-mail: vasques@fe.up.pt

Abstract

Fieldbus communication networks aim to interconnect sensors, actuators and controllers within distributed computer-controlled systems. Therefore, they constitute the foundation upon which real-time applications are to be implemented. A specific class of fieldbus communication networks is based on a simplified version of token-passing protocols, where each station may transfer, at most, one Single Message per Token Visit (SMTV).

In this paper, we establish an analogy between non pre-emptive task scheduling in single processors and scheduling of messages on SMTV token-passing networks. Moreover, we clearly show that concepts such as blocking and interference in non pre-emptive task scheduling have their counterparts in the scheduling of messages on SMTV token-passing networks. Based on this task/message scheduling analogy, we provide pre-run-time schedulability conditions for supporting real-time messages with SMTV token-passing networks. We provide both utilisation-based and response time tests to perform the pre-run-time schedulability analysis of real-time messages on SMTV token-passing networks, considering RM/DM and EDF priority assignment schemes.

1. Introduction

Fieldbus networks are widely used as the communication support for distributed computer-controlled systems (DCCS), in applications ranging from process control to discrete manufacturing. Usually, DCCS impose real-time requirements; that is, traffic must be sent and received within a bounded interval, otherwise a timing fault is said to occur. This motivates the use of communication networks within which the medium access control (MAC) protocol is able to schedule messages according to their real-time requirements.

One of the MAC protocols used in widely accepted fieldbus networks is based on a simplified version of the token-passing protocol. In these fieldbus networks (PROFIBUS and P-NET are just two examples) there are typically two types of network nodes: masters and slaves. This MAC protocol is based on a token-passing procedure used by master stations to grant the bus access to each other, and a master-slave procedure used by master stations to communicate with slave stations. Each time a master receives the token, it will be able to perform, at most, one message cycle. We denote this type of networks as Single Message per Token Visit (SMTV) token-passing networks.

A message cycle consists of a master's action frame (request or send/request frame) and the associated responder's acknowledgement or response frame. We assume that responses from slaves are immediate, with a bounded turnaround time. At each master node, requests generated at the application process (AP) level are placed in the master's outgoing queue. At each token arrival, the highest priority message cycle will be processed. We assume the following message stream model:

$$S_i^k = (C_i^k, T_i^k, D_i^k) \quad (1)$$

S_i^k defines a message stream i in master k (k = 1, ..., n). A message stream is a temporal sequence of message cycles concerning, for instance, the remote reading of a specific process variable. C_i^k is the longest message cycle duration of stream S_i^k. This duration includes both the longest request and response transmission times, and also the worst-case slave's turnaround time. T_i^k is the periodicity of stream S_i^k requests. In order to have a subsequent timing analysis independent from the model of the tasks at the application process level, we assume that this periodicity is the minimum interval between two consecutive arrivals of S_i^k requests to the outgoing queue. Finally, D_i^k is the relative deadline of a message cycle; that is, the maximum admissible time interval between the instant when the message request is placed in the outgoing queue and the instant at which the related response is completely received at the master's incoming queue. Finally, ns^k denotes the number of message streams associated with a master k.

In our model, the relative deadline of a message can be equal or smaller than its period ($D_i^k \leq T_i^k$). Thus, if in the outgoing queue there are two message requests from the same message stream, this means that a deadline for the first of the requests was missed. Therefore, the maximum number of pending requests in the outgoing queue will be ns^k.

We denote the worst-case response time of a message stream S_i^k as R_i^k. This time is measured starting at the instant when the request is placed in the outgoing queue, until the instant when the response is completely received at the incoming queue. Basically, this time span is made up of two components. One concerns the time spent by the request in the outgoing queue, until gaining access to the

0-7695-0734-4/00 $10.00 © 2000 IEEE

bus (queuing delay). The other concerns the time needed to process the message cycle, that is, to send the request and receive the related response (transmission delay). Thus,

$$R_i^k = Q_i^k + C_i^k \quad (2)$$

where Q_i^k is the worst-case queuing delay of a message request from S_i^k.

Analysis for the worst-case response time can be performed if the worst-case token rotation time is assumed for all token cycles. Assume also that C_M is the maximum transmission duration of a message cycle. If a master uses the token to perform a message cycle, we can define the token holding time as:

$$H = r + C_M + t \quad (3)$$

where the symbol t denotes the time to pass the token after a message cycle has been performed and the symbol r denotes the master's worst-case reaction time. As the token rotation time is the time interval between two consecutive token visits, the worst-case token rotation time, denoted as V, is:

$$V = n \times H \quad (4)$$

Considering priority-based dispatching mechanisms, the worst-case response time for a message request occurs when the request is placed in the master's queue just after the token arrival, hence not being able to be processed in that token visit. If there were any other message request pending before the token arrival, then the token would have been used to transmit that message; otherwise, the master would not use the token at all. Therefore, the worst-case response time of a message stream S_i^k will be:

$$R_i^k = Q_i^k + C_i^k = V + \Phi \times V + C_i^k \quad (5)$$

where the first term V denotes the message blocking, and the symbol Φ denotes the number of higher-priority messages (interference) that can be scheduled ahead of a message from S_i^k. Fig. 1 illustrates the case where Φ equals 0 (highest priority message cycle S_1^1).

The remainder of this paper is organised as follows. In Section 2 we survey some relevant results for the priority-based schedulability analysis of real-time tasks, both for fixed and dynamic priority assignment schemes. We give emphasis to the worst-case response time analysis in non pre-emptive contexts, since that analysis is of paramount importance to the message schedulability analysis in SMTV token-passing communication networks. In Section 3 we discuss the analogy between task scheduling in a single processor environment and message scheduling in SMTV token-passing networks. Based on this task/message scheduling analogy we provide utilisation-based tests and response time tests, in Section 4 and Section 5, respectively, for SMTV token-passing networks. In both cases, RM/DM and EDF priority assignment schemes are considered. Finally, in Section 6, some conclusions are drawn.

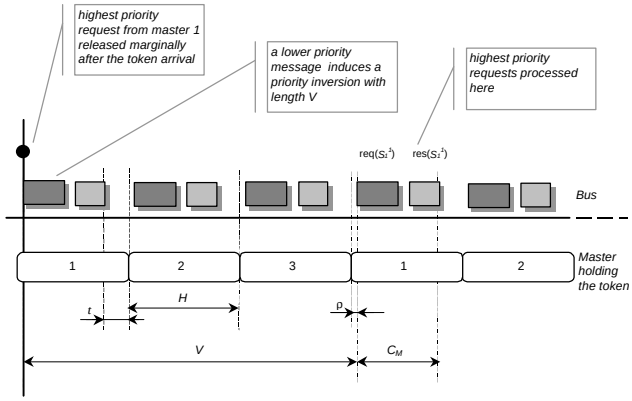

Figure 1

2. Schedulability Analysis of Tasks

Real-time computing systems with tasks dispatched according to a priority-based policy (we consider only RM/DM or EDF), must be tested a-priori in order to check if, during run time, no deadline will be missed. This feasibility test is called the pre-run-time schedulability analysis of the task set.

There are mainly two types of analytical methods to perform pre-run-time schedulability analysis. One is based on the analysis of the processor utilisation. The other is based on the response time analysis for each individual task. In [1], the authors demonstrated that by considering only the processor utilisation, a test for the pre-run-time schedulability analysis could be obtained. Contrarily, a response time test must be performed in two stages. First, an analytical approach is used to predict the worst-case response time of each task. The values obtained are then compared, trivially, with the relative deadlines of the tasks.

The utilisation-based tests have a major advantage: it is a simple computation procedure, which is applied to the overall task set. By this reason, they are very useful for implementing schedulers that check the schedulability online. However, utilisation-based tests have also important drawbacks, when compared with their response-time counterparts. They do not give any indication of the actual response times of the tasks. More importantly, and apart from particular task sets, they constitute sufficient but not necessary conditions. This means that if the task set passes the test, the schedule will meet all deadlines, but if it fails the test, the schedule may or may not fail at run-time (hence, there is a certain level of pessimism). It is also worth mentioning that the utilisation-based tests cannot be used for more complicated task models [2].

2.1. Schedulability Tests: Fixed Priorities

2.1.1. Basic Utilisation-Based Test

For the RM priority assignment, Liu and Layland introduced an utilisation-based sufficient test:

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \le N \times \left(2^{1/N} - 1\right) \quad (6)$$

This utilisation-based test is valid for periodic independent tasks, with relative deadlines equal to the period, and for pre-emptive systems. In [3], the authors provide an exact analysis:

$$\min_{(k,l)\in R_i} \left\{ \sum_{j=1}^{i} \left(U_j \times \frac{T_j}{l \times T_k} \times \left\lceil \frac{l \times T_k}{T_j} \right\rceil \right) \right\} \le 1, \quad \forall_{i,\ 1 \le i \le n} \quad (7)$$

where $R_i = \{(k,l)\}$ with $1 \le k \le i$ and $l = 1, ..., \lfloor T_i/T_k \rfloor$. It is clear that inequality (7) is not an easy to use test, hence loosing one of the advantages inherent to the more basic formulations: its simplicity.

2.1.2. Extended Utilisation-Based Test

In [4], the authors update the utilisation-based test (6) to include blocking periods, during which higher-priority tasks are blocked by lower-priority ones, to solve the problem of non-independence of tasks:

$$\left(\sum_{i=1}^{i} \frac{C_i}{T_i} \right) + \frac{B_i}{T_i} \le i \times \left(2^{1/i} - 1\right), \quad \forall_{i,\ 1 \le i \le N} \quad (8)$$

where B_i is the maximum blocking a task i can suffer.

This analysis can be extended to a non pre-emptive context, since in this case a higher-priority task can also be "blocked" by a lower-priority task. Assuming that the tasks are completely independent, the maximum blocking time a task can suffer is given by:

$$\begin{cases} B_i = 0, & \text{if } P_i = \min\limits_{j=1,..,N} \{P_j\} \\ B_i = \max\limits_{j \in lp(i)} \{C_j\}, & \text{if } P_i \ne \min\limits_{j=1,..,N} \{P_j\} \end{cases} \quad (9)$$

where $lp(i)$ denotes the set of lower-priority tasks (than task i). Therefore, inequality (8) can be used as an utilisation-based test for a set of non pre-emptable independent tasks, with the blocking for each task as given by (9). Moreover, accepting an increased level of pessimism, inequality (8) can be simplified to:

$$\left(\sum_{i=1}^{N} \frac{C_i}{T_i} \right) + \max_{i,\ 1 \le i \le N} \left\{ \frac{B_i}{T_i} \right\} \le i \times \left(2^{1/N} - 1\right) \quad (10)$$

Note that if all tasks have the same computation time, (10) considers that each task may be blocked at the rate of the highest-priority task.

2.1.3. Response Time Tests: Pre-emptive Context

In [5] the authors proved that the worst-case response time R_i of a task i is found when all tasks are synchronously released (critical instant) at their maximum rate. R_i is defined as:

$$R_i = I_i + C_i \quad (11)$$

where I_i is the maximum interference that task i can experience from higher-priority tasks in any interval $[t, t + R_i)$. Without loss of generality, it can be assumed that all processes are released at time instant 0. Consider a task j with higher-priority than task i. Within the interval $[0, R_i)$, it will be released $\lceil R_i/T_j \rceil$ times. Therefore, each release of task j will impose an interference C_j. The worst-case response time R_i of a task t_i is then:

$$R_i = \sum_{j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil \times C_j \right) + C_i \quad (12)$$

where $hp(i)$ denotes the set of higher-priority tasks (than task i). Equation (12) embodies a mutual dependence, since R_i appears in both sides of the equation. The easiest way to solve such equation is to form a recurrence relationship [6]:

$$W_i^{m+1} = \sum_{j \in hp(i)} \left(\left\lceil \frac{W_i^m}{T_j} \right\rceil \times C_j \right) + C_i \quad (13)$$

The recursion ends when $W_i^{m+1} = W_i^m = R_i$ and can be solved by successive iterations starting from $W_i^0 = C_i$. Indeed, it is easy to show that W_i^m is non-decreasing. Consequently, the series either converges or exceeds T_i (case of RM) or D_i (case of DM). If the series exceeds T_i (or D_i), the task t_i is not schedulable.

2.1.4. Response Time Tests: non Pre-emptive Context

In [6] the authors updated the analysis of [5] to include blocking factors introduced by periods of non pre-emption, due to the non-independence of the tasks. The worst-case response time is updated to:

$$R_i = B_i + \sum_{j \in hp(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil \times C_j \right) + C_i \quad (14)$$

which may also be solved using a similar recurrence relationship. B_i is also as given by equation (9).

Some care must be taken using equation (14) for the evaluation of the worst-case response time of non pre-emptable independent tasks. In the case of pre-emptable tasks, with equation (12) we are finding the processor's level-i busy period preceding the completion of task i; that is, the time during which task i and all other tasks with a priority level higher than the priority level of task i still have processing remaining. For the case of non pre-emptive tasks, there is a slight difference, since for the evaluation of the processor's level-i busy period we cannot include task i itself; that is, we must seek the time instant preceding the execution start time of task i.

Therefore, equation (11) can be used to evaluate the task's response time of a task set in a non pre-emptable context and independent tasks, where the interference is:

$$I_i = B_i + \sum_{j \in hp(i)} \left(\left\lceil \frac{I_i}{T_j} \right\rceil \times C_j \right) \quad (15)$$

Note also that a re-definition for the critical instant must be made. The maximum interference occurs when task *i* and all other higher-priority tasks are synchronously released just after the release of the longest lower-priority task (than task *i*).

2.2. Schedulability Tests: Dynamic Priorities

2.2.1. Basic Utilisation-Based Test

For the EDF priority assignment, Liu and Layland also introduced an utilisation-based pre-run-time schedulability test (16), valid for non pre-emptive, independent and periodic tasks, for which the relative-deadline is equal to the period.

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1 \tag{16}$$

Inequality (16) can be easily updated to include blocking periods due to the non-independence of the tasks. In [7], the author updated inequality (16) to:

$$\left(\sum_{i=1}^{i} \frac{C_i}{T_i} \right) + \frac{B_i}{T_i} \leq 1, \quad \forall_{i,\, 1 \leq i \leq N} \tag{17}$$

where B_i is the maximum blocking a task *i* can suffer, considering the stack resource protocol (SRP). The key idea behind the SRP is that when a job needs a resource which is not available, it is blocked at the time it attempts to pre-empt, rather than later. This makes inequality (17) valid for sets of non pre-emptable tasks, dispatched according to the EDF scheme.

Similarly to the updating of (8) to (10), (17) can be updated to a simpler (but more pessimistic) test:

$$\left(\sum_{i=1}^{N} \frac{C_i}{T_i} \right) + \max_{i,\, 1 \leq i \leq N} \left\{ \frac{B_i}{T_i} \right\} \leq 1 \tag{18}$$

where B_i is defined as:

$$B_i = \max_{j \neq i} \{ C_j \} \tag{19}$$

Another relevant result from [7] is that (17) can also be extended to task sets with relative deadlines smaller than periods:

$$\left(\sum_{i=1}^{i} \frac{C_i}{D_i} \right) + \frac{B_i}{D_i} \leq 1, \quad \forall_{i,\, 1 \leq i \leq N} \tag{20}$$

As a corollary, inequality (16) can be extended for task sets with $D_i \leq T_i$:

$$\sum_{i=1}^{N} \frac{C_i}{D_i} \leq 1 \tag{21}$$

These simple utilisation-based tests ((18) and (20)) are however quite pessimistic. Less pessimistic utilisation-based tests will now be addressed in Sections 2.2.2 and 2.2.3, for pre-emptive and non pre-emptive tasks, respectively. Later, in Sections 2.2.4 and 2.2.5, recent results on response time analysis will be addressed, for pre-emptive and non pre-emptive tasks, respectively.

2.2.2. Extended Utilisation Tests: Pre-emptive Context

In [8] the author extends the results of Liu and Layland in order to consider sporadic tasks, where inequality (16) is updated to:

$$\sum_{i=1}^{N} \left\lceil \frac{t - D_i}{T_i} \right\rceil^{+} \times C_i \leq t, \ \forall_{t \geq 0} \tag{22}$$

with $\lceil x \rceil^{+} = 0$ if $x < 0$. This formulation has advantages over (20), in the sense that it turns out to be a necessary and sufficient condition. Inequality (22) can not be classified as a simple test when compared to (20). It has also an additional problem, since it must be checked over an infinite continuous time interval $[0, \infty)$. A simplification to the schedulability test can be made considering that the right side of inequality (22) does only change at $k \times T_i + D_i$ time instants, and thus the inequality does only need to be checked for these time instants. Different authors have addressed the problem of finding an upper limit for *t*. It is possible to prove that if the total utilisation of the processor is ≤ 1 (condition (16)), it exists a point $t_{\max}$, such that $\sum_{i=1,\ldots,n} (\lceil (t - D_i)/T_i \rceil^{+} \times C_i) \leq t$ always hold, $\forall_{t \geq t_{\max}}$. Consequentely, inequality (22) can be re-written as follows:

$$\sum_{i=1}^{N} \left\lceil \frac{t - D_i}{T_i} \right\rceil^{+} \times C_i \leq t, \ \forall_{t \in S}, \quad \text{with } S = \left(\bigcup_{i=1}^{N} \{ D_i + k \times T_i, k \in \aleph \} \right) \cap [0, t_{\max}) \tag{23}$$

In [9] the authors demonstrated that t_{max} could be given by $(U/(1\text{-}U)) \times \max_{i=1,\ldots,N} \{(Ti\text{-}Di)\}$, where *U* represents the overall processor's utilisation ($\sum_{i=1,\ldots,N} (C_i/T_i)$). This result was further improved in [10], where the upper bound for *t* is defined as $t_{max} = ((\sum_{i=1,\ldots,N} (1\text{-}D_i/T_i) \times C_i)/(1\text{-}U)$. Although this last formulation gives a smaller value for t_{max}, it still suffers from the same disadvantage: as the overall utilisation approaches 1, its value becomes very large.

Another approach is considered in [10] and [11], where the authors demonstrate that $t_{max} = L$ (synchronous processor's busy period). The synchronous processor's busy period is defined as the time interval from the critical instant up to the first instant when there are no more pending tasks in the system:

$$L = \sum_{i=1}^{N} \left\lceil \frac{L}{T_i} \right\rceil \times C_i \tag{24}$$

Equation (24) may be solved by recurrence, starting with $L^0 = \sum_{i=1,\ldots,N} C_i$.

2.2.3. Extended Utilis. Tests: non Pre-emptive Context

For the non pre-emptive context, a similar test was presented in [8] and [12]:

$$\sum_{i=1}^{N}\left\lceil\frac{t-D_i}{T_i}\right\rceil^{+}\times C_i+\max_{j=1,\ldots,N}\{C_j\}\le t,\ \forall_{t\ge D_{\min}},\quad \text{with } D_{\min}=\min_{j=1,\ldots,N}\{D_j\} \tag{25}$$

Comparing to the test for the pre-emptive context (22), the inclusion of the blocking factor is intuitive (see Section 2.2.1.). However, in [13] the authors discuss the pessimism inherent to the inequality (25). The main argument is that in this inequality it is considered that the cost of possible priority inversions is always initiated by the longest task and, moreover, it is effective during the entire interval under analysis. To reduce this level of pessimism, it is suggested the following modification:

$$\sum_{i=1}^{N}\left\lceil\frac{t-D_i}{T_i}\right\rceil^{+}\times C_i+\max_{\substack{j=1,\ldots,N\\ D_j>t}}\{C_j\}\le t,\ \forall_{t\in S},\quad \text{with } \max_{\substack{j=1,\ldots,N\\ D_j>t}}\{C_j\}=0 \text{ if } \nexists_j : D_j>t \tag{26}$$

That is, the blocking factor is only included if its deadline occurs after *t*.

2.2.4. Response Time Tests: Pre-emptive Context

The worst-case response time analysis for pre-emptive EDF scheduling was first introduced in [14]. In his work, Spuri demonstrated that the worst-case response time of a task *i* is found in the processor's deadline-*i* busy period (analogous to the processor's level-*i* busy period in the case of fixed priorities). However, the longest processor's deadline-*i* busy period may occur when all tasks but task *i* (contrarily to the case of fixed priority assignment) are synchronously released and at their maximum rate. This means that, in order to find the worst-case response time of task *i*, we need to examine multiple scenarios within which, while task *i* has an instance released at time *a*, all other tasks are synchronously released at time *t* = 0. Thus, given a value of *a*, the response of an instance of task *i* is:

$$R_i(a)=\max\{C_i, L_i(a)-a\} \tag{27}$$

where $L_i(a)$ is the length of the deadline-*i* busy period, which starts at time instant *t* = 0. $L_i(a)$ can be evaluated by the following iterative computation:

$$L_i(a)=\sum_{\substack{j\ne i\\ D_j\le a+D_i}}\left(\min\left\{\left\lceil\frac{L_i(a)}{T_j}\right\rceil, 1+\left\lfloor\frac{a+D_i-D_j}{T_j}\right\rfloor\right\}\times C_j\right)+\left(1+\left\lfloor\frac{a}{T_i}\right\rfloor\right)\times C_i \tag{28}$$

Equation (28) can be solved by recurrence, starting with $L_i^0(a) = 0$. Obviously, in equation (28), the computational load only considers tasks that have deadlines earlier than D_i. Finally, in the general case, the worst-case response time for a given task *i* is:

$$R_i=\max_{a\ge 0}\{R_i(a)\} \tag{29}$$

The remaining problem is how to determine the values of *a*. Looking to the right-hand side of equation (28), we can easily understand that its value only changes at $k\times T_j + D_j - D_i$ steps.

$$a\in\bigcup_{j=1}^{N}\{k\times T_j+D_j-D_i, k\in\aleph_0\}\cap[0, L[\tag{30}$$

with *L* as given by equation (24).

2.2.5. Response Time Tests: non Pre-emptive Context

The worst-case response time analysis for the non pre-emptive EDF scheduling was introduced in [13]. The main difference from the analysis for the pre-emptive case is that a task instance with a later absolute deadline can cause a priority inversion. Thus, and similarly to the fixed priority case (Section 2.2.4), instead of analysing the deadline-*i* busy period preceding the completion time of task *i*, we must analyse the busy period preceding the execution start time of the task's instance. Consequently, the response time of the t_i 's instance released at time *a* is:

$$R_i(a)=\max\{C_i, L_i(a)+C_i-a\} \tag{31}$$

where $L_i(a)$ is now the length of the busy period (preceding execution). Thus, $R_i(a)$ can be evaluated by means of the following iterative computation:

$$L_i(a)=\max_{D_j>a+D_i}\{C_j\}+\sum_{\substack{j\ne i\\ D_j\le a+D_i}}\left(\min\left\{1+\left\lfloor\frac{L_i(a)}{T_j}\right\rfloor, 1+\left\lfloor\frac{a+D_i-D_j}{T_j}\right\rfloor\right\}\times C_j\right)+\left\lfloor\frac{a}{T_i}\right\rfloor\times C_i \tag{32}$$

3. Analogies to Message Scheduling in SMTV

In this section we discuss the analogy between task scheduling in a single processor environment and message scheduling in SMTV token-passing networks. This analogy will later enable the formulation of feasibility tests for the pre-run-time schedulability analysis of message stream sets in SMTV token-passing networks.

Table 1

Task	*Computation Time (C)*	*Period (T)*
A	10	60
B	10	80
C	10	100
D	10	100

In the schedulability analysis of tasks in the non pre-emptive context, the concept of processor's busy period denotes the time interval within which the processor is not idle (see Section 2.1.4). Consider the task set example (*D*=*T*) of Table 1.

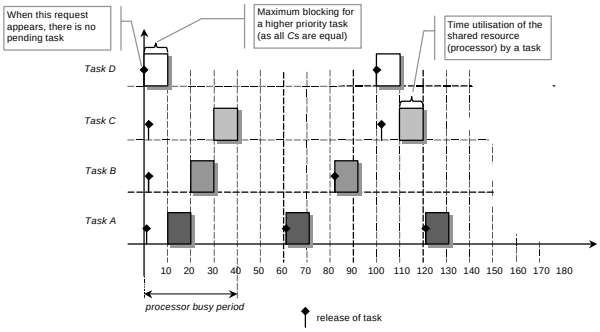

Figure 2

Assuming a RM priority assignment policy in a non pre-emptive context, Fig. 2 illustrates a time-line considering that the first instance of task *D* (lower-priority task) is released marginally after time instant 0, and before all other instances of higher-priority tasks. Note that the blocking of a task in a non pre-emptive context is equal to the maximum execution length of a lower-priority task (see equation (9)).

Consider now the following message stream set example (where $D=T$) to be scheduled in a SMTV token-passing network:

Table 2

Message	*Message Cycle Length (C)*	*Period (T)*
A	2	60
B	2	80
C	2	100
D	2	100

This case will be shown to be loosely equivalent to the previous task scheduling example, when the token cycle time is equal to the tasks' execution time ($V = C_{task}$).

Consider now Fig. 3, which illustrates the time-line for a message scheduling on a SMTV token-passing network, considering a messages' release pattern (arrival of requests to the outgoing queue) similar to the previous tasks' release pattern. It is clear that the message blocking time is equal to the token cycle time. However, this blocking term is independent of the priority ordering of message transfers. Therefore, the blocking problem in the task scheduling theory can only be considered to be loosely equivalent to the blocking problem in SMTV token-passing networks, since the priority ordering property is not preserved.

It is also clear that tests available for the schedulability analysis of non pre-emptable tasks in single processor systems can be adapted to the message scheduling in SMTV token-passing networks, considering that the blocking term is equal to the token cycle time, independently of the message priority.

Therefore, the computation time of a task can be considered equivalent to the token cycle time, since in a SMTV token-passing network the shared resource (network access/token) is available once in every *V* interval. This means that the contribution of each higher-priority message cycle to the overall queuing delay of a lower-priority message cycle is always equal to *V*.

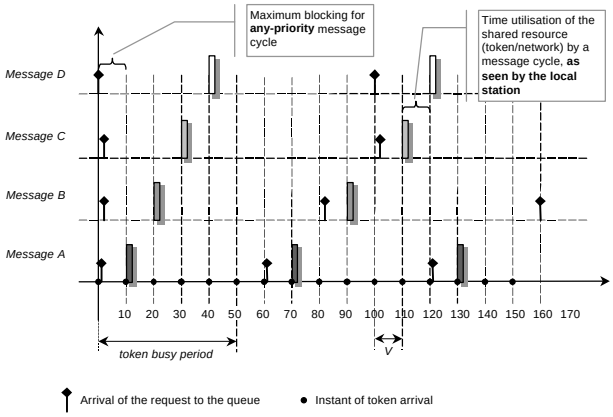

Figure 3

Finally, in Table 3 we summarise the analogies between task scheduling in non pre-emptive single processor environments and message scheduling in SMTV token-passing networks.

Table 3

	Task Scheduling	*Message Scheduling*
Maximum blocking (except for the lowest priority task/message)	$B_i = \max_{\forall j \in lp(i)} \{C_j\}$	V
Maximum blocking (lowest priority task/message)	0	V
Resource usage time for the higher-priority tasks/messages	C_j	V
Resource usage time for the task/message itself	C_i	C_i

Considering these analogies between task scheduling and message scheduling, a set of (token) utilisation-based and response time tests can be developed for the schedulability analysis of SMTV token-passing networks. In the following sections we present both types of tests.

4. (Token) Utilisation-Based Tests

In this section, we derive utilisation-based schedulability tests for both fixed and dynamic priority assignment schemes. Such schedulability tests, which can be quite pessimistic, provide a tool to evaluate the schedulability of the overall message set with a reduced complexity.

4.1. Case of Rate Monotonic Priority Assignment

Considering the analogies to the blocking and tasks' computation time drawn in the previous section, the schedulability test (10) for the RM dispatched tasks can be adapted to encompass the characteristics of the SMTV token-passing protocol, as follows:

$$\left(\sum_{i=1}^{ns^k} \frac{V}{T_i^k} \right) + \max_{1 \le i \le n} \left\{ \frac{V}{T_i^k} \right\} \le ns^k \times \left(2^{\frac{1}{ns^k}} - 1 \right), \ \forall_k \qquad (33)$$

As the worst-case token cycle time (V) is constant, equation (33) can be re-written as:

$$V \times \left[\left(\sum_{i=1}^{ns^k} \frac{1}{T_i^k} \right) + \frac{1}{\min\{T_i^k\}} \right] \leq ns^k \times \left(2^{\frac{1}{ns^k}} - 1 \right), \ \forall_k \qquad (34)$$

Note that, as we are considering SMTV token-passing networks, the interference from other masters is only reflected on the evaluation of the V parameter.

Consider the following example, which highlights the use of the proposed schedulability test (34):

Table 4

Stream	*Period*
S_1^k	5
S_2^k	7
S_3^k	8
S_4^k	12

Considering that the worst-case token rotation time is V = 1, it follows that the schedulability test is:

$$1 \times \left[\left(\sum_{i=1}^{4} \frac{1}{T_i^k} \right) + \frac{1}{5} \right] \leq 4 \times \left(2^{\frac{1}{4}} - 1 \right) \Leftrightarrow 0.75 \leq 0.76$$

Therefore the message stream set of Table 4 is schedulable by the RM algorithm in a SMTV token-passing network. In Fig. 4, we present a possible time-line for the message scheduling, assuming that all messages are requested just after the first token arrival. In this way, we represent a blocking term at the beginning of the time-line.

This example highlights the pessimism associated to the utilisation-based tests, since, although the schedulability test is just marginally true, none of the message cycles is scheduled close to its deadline.

4.2. Case of EDF Priority Assignment

Considering again the analogies to the blocking and tasks' computation time drawn in Section 3, the schedulability test (18) for the EDF dispatched tasks can also be adapted to encompass the characteristics of the SMTV token-passing protocols, as follows:

$$V \times \left[\left(\sum_{i=1}^{ns^k} \frac{1}{T_i^k} \right) + \frac{1}{\min\limits_{1 \leq i \leq n} \{T_i^k\}} \right] \leq 1, \ \forall_k \qquad (35)$$

Consider now the example of Table 5, where periods are considered to be marginally smaller than multiples of the worst-case token cycle time (V = 1). The application of the schedulability test (35) to this message stream set is:

$$1 \times \left[\left(\sum_{i=1}^{4} \frac{1}{T_i^k} \right) + \frac{1}{4} \right] \leq 1 \Leftrightarrow 0.99 \leq 1 \ \text{TRUE}$$

Hence, this message stream set is schedulable considering the EDF priority assignment scheme, while with the RM assignment scheme would not pass the schedulability test (34): 0.99 ≤ 0.76 is FALSE.

Table 5

Stream	*Period*
S_1^k	4
S_2^k	5
S_3^k	6
S_4^k	8

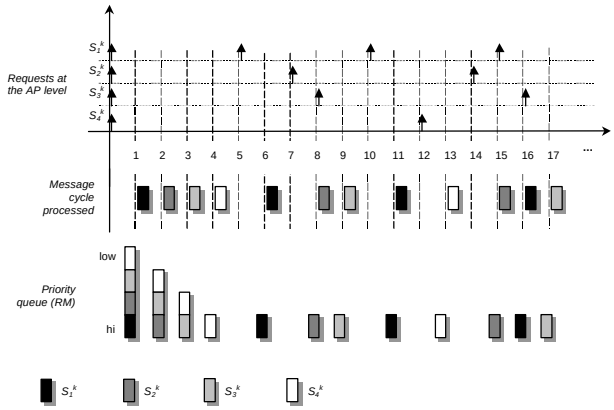

Figure 4

5. Response Time Tests

In this section we derive response time schedulability tests for both fixed and dynamic priority assignment schemes. Such schedulability tests, compared to the (token) utilisation-based tests are more complex, but also much less pessimistic, as it will be shown in this section. This is an expected result, as response time tests for task scheduling are sufficient and necessary conditions, while the utilisation-based tests are generally only sufficient conditions (refer to Section 2). It is also important to note that for the case of $D_i^k < T_i^k$ there are no simple utilisation-based tests for the case of fixed priorities.

5.1. Response Time Tests: Fixed Priority

Based on the analogies between task scheduling and message scheduling on SMTV token-passing networks, the worst-case response time analysis for the non pre-emptive context (refer to Section 2.1.4) can be adapted to encompass the characteristics of the SMTV token-passing protocols. The worst-case message response is defined in equation (2), where Q_i^k is:

$$Q_i^k = V + \sum_{\forall_{j \in hp(i)}} \left\lceil \frac{Q_i^k}{T_j^k} \right\rceil \times V = V \times \left(1 + \sum_{\forall_{j \in hp(i)}} \left\lceil \frac{Q_i^k}{T_j^k} \right\rceil \right) \qquad (36)$$

Note that this queuing delay is the equivalent to the task's interference in a non pre-emptive context (15).

Considering again the message stream set example of Table 5, the worst-case response time for each message stream will be as shown in Table 6 (with $C_i^k = 0.2, \forall_i$).

Table 6

Stream	*Response*
S_1^k	1.2
S_2^k	2.2
S_3^k	3.2
S_4^k	7.2

Considering the message stream S_4^k, the iterations for evaluating its queuing delay are as follows:

$$Q_4^{k(0)} = 1; Q_4^{k(1)} = 4; Q_4^{k(2)} = 5; Q_4^{k(3)} = 6; Q_4^{k(4)} = 7; Q_4^{k(5)} = 7$$

and iterations stop, since $Q_4^{k(5)} = Q_4^{k(4)} = 7$. Therefore, from equation (2) $R_4^k = 7 + 0.2 = 7.2$, which is smaller than its relative deadline (its period), and thus, the message stream set is RM schedulable. Considering the same message stream set, the (token) utilisation-based test (34) gives $0.99 \leq 0.76$, which is equivalent to state that this message stream set may or not be schedulable. Therefore, it turns out that the response time test is much less pessimistic than the (token) utilisation-based test. The time-line presented in Fig. 5 illustrates the above results.

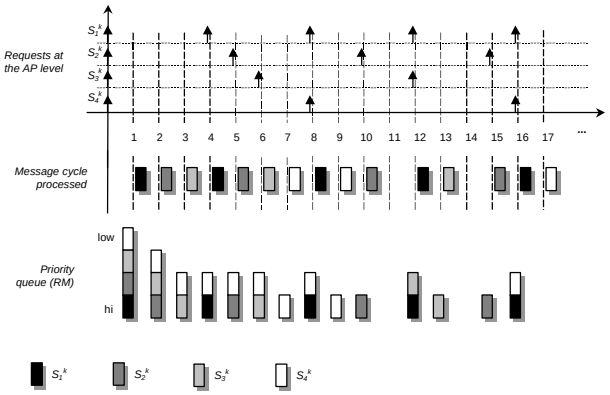

Figure 5

5.2. Response Time Tests: Dynamic Priority

Based on the analogies between task scheduling and message scheduling on SMTV token-passing networks, the worst-case response time analysis for the non pre-emptive context (refer to Section 2.2.5) can also be adapted to encompass the characteristics of the SMTV token-passing protocols. The worst-case message response time is, obviously, given by equation (2). However, a major difference exists for the definition of the queuing delay, which for the EDF case must be defined as:

$$Q_i^k = V \times \left(1 + \sum_{\substack{j \neq i \\ D_j^k \leq D_i^k}} \min \left\{ 1 + \left\lfloor \frac{Q_i^k}{T_j^k} \right\rfloor, 1 + \left\lfloor \frac{D_i^k - D_j^k}{T_j^k} \right\rfloor \right\} \right) \quad (37)$$

that is, a message request concerning stream S_i^k will be delayed by all message requests of other streams having earlier or equal absolute deadlines than the absolute deadline for S_i^k (absolute deadlines are the difference between the relative deadline, D_i^k, and the beginning of the evaluation interval - assumed at time instant 0). Note that while $\sum(1 + \lfloor Q_i^k / T_j^k \rfloor)$ requests having relative deadlines smaller or equal to D_i^k can be placed in the AP queue, from those requests, only a maximum of $1 + \lfloor (D_i^k - D_j^k)/ T_j^k \rfloor$ will have absolute deadlines earlier than D_i^k. We illustrate this effect with the following example.

Table 7

Stream	*Period*
S_1^k	4
S_2^k	5
S_3^k	6
S_4^k	7

If we consider the synchronous release pattern for message streams (Table 7), a time-line for the EDF schedule may be as illustrated in Fig. 6.

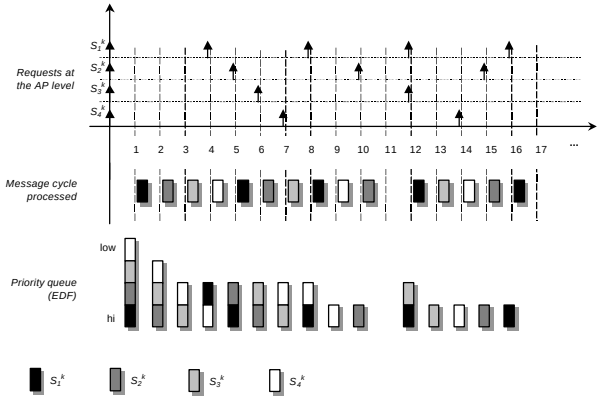

Figure 6

As it can be seen from Fig. 6, there is a request for S_1^k arriving to the queue before the processing of the first request for S_4^k. However, as that request for S_1^k has an absolute deadline which is later than the absolute deadline for S_4^k, it will be processed only after the request for S_4^k. This behaviour of the EDF scheduler is effectively translated by equation (37), as can be seen by the following successive iterations ($V = 1$):

$$Q_4^{k(0)} = 1 + \min\{1,1\} + \min\{1,1\} + \min\{1,1\} = 4$$
$$Q_4^{k(1)} = 1 + \min\{2,1\} + \min\{1,1\} + \min\{1,1\} = 4$$

and iterations stop, as $Q_4^{k(1)} = Q_4^{k(0)} = 4$. The maximum queuing delay for a request of stream S_4^k, considering that the streams have a synchronous release pattern, is thus as shown in Fig. 6.

Note however that the worst-case response time for EDF dispatched messages is not necessarily found with a

synchronous release pattern (refer to Sections 2.2.4 and 2.2.5). Therefore, equation (37) must be updated to:

$$Q_i^k(a) = B_i^k(a) + V \times \left(\sum_{\substack{j \neq i \\ D_j^k \leq a + D_i^k}} \left(\min\left\{ 1 + \left\lfloor \frac{Q_i^k(a)}{T_j^k} \right\rfloor, 1 + \left\lfloor \frac{a + D_i^k - D_j^k}{T_j^k} \right\rfloor \right\} \right) + \left\lfloor \frac{a}{T_i^k} \right\rfloor \right) \quad (38)$$

where B_i^k is defined as follows:

$$B_i^k(a) = \begin{cases} V, & a = 0 \\ V, & a \neq 0 \ \wedge \ \exists_j : D_j^k > a + D_i^k \end{cases} \quad (39)$$

Note that while with the RM/DM approach (Section 5.1) the blocking term is V and effective for all the message streams, with the EDF approach, we must only consider (if $a \neq 0$) a blocking if it exists a message stream S_j^k ($j \neq i$) with an absolute deadline later than the relative deadline of the instance of S_i^k released at time instant a. A main difference exists in comparison to the analogous formulation for task scheduling (32), since in the case of the SMTV token-passing model, for $a = 0$ there is always a blocking with the value V. Similarly to the case of task scheduling, a belongs to the following set of values:

$$a \in \left\{ 0, \left\{ \bigcup_{l=1}^{ns^k} \left\{ \Psi \times T_l^k + D_l^k - D_i^k, \ \Psi \in \aleph_0 \right\} \cap [0, L[\right\} \right\} \quad (40)$$

where the (token) synchronous busy period is given by:

$$L = \sum_{i=1}^{ns^k} \left\lceil \frac{L}{T_i^k} \right\rceil \times V \quad (41)$$

The queuing delay is thus:

$$Q_i^k = \max_a \left\{ 0, Q_i^k(a) - a \right\} \quad (42)$$

since the computation $Q_i^k(a)$ may occasionally give a value smaller than a (for instance, when the value of a corresponds to more than one request of S_i^k during the interval under analysis, the interval $[0, Q_i^k(a)]$.

Finally, substituting equation (42) back in equation (2), the worst-case response time of a message stream dispatched according to the EDF scheme is:

$$R_i^k = \max_a \left\{ 0, Q_i^k(a) - a \right\} + C_i^k \quad (43)$$

The analysis outlined will be now illustrated for the stream set example of table 7. The results presented were obtained using the following exact characterisation of the message stream set of table 7:

Table 8

Stream	C_i^k	T_i^k	D_i^k
S_1^k	0.2	3.99	3.99
S_2^k	0.2	4.99	4.99
S_3^k	0.2	5.99	5.99
S_4^k	0.2	6.99	6.99

For this message stream set, the value for L (upper bound for a) is: $L = 9$. Therefore, the values of a that must be tested for each message stream are:

Table 9

Stream	*a* = 0	*a1*	*a2*	*a3*	*a4*	*a5*	*a6*	*a7*
S_1^k	0.00	1.00	2.00	3.00	3.99	5.99	7.98	7.99
S_2^k	0.00	1.00	2.00	2.99	4.99	6.98	6.99	8.99
S_3^k	0.00	1.00	1.99	3.99	5.98	5.99	7.99	8.98
S_4^k	0.00	0.99	2.99	4.98	4.99	6.99	7.98	8.97

In order to evaluate the queuing delay for each release pattern, equation (7.8) must be evaluated for each a value. The results for ($Q_i^k(a)$ - a) are:

Table 10

Stream	*a* = 0	*a1*	*a2*	*a3*	*a4*	*a5*	*a6*	*a7*
S_1^k	1.00	1.00	1.00	0.00	-0.99	1.01	-0.98	0.01
S_2^k	2.00	2.00	1.00	0.01	-1.99	0.02	2.01	1.01
S_3^k	3.00	2.00	1.01	-0.99	-2.98	-2.99	2.01	2.02
S_4^k	4.00	2.01	0.01	-1.98	-1.99	-3.99	3.02	2.03

In this table, for each message stream the value of $\max\{0, Q_i^k(a) - a\}$ is highlighted. The worst-case response times for the message streams are presented in Table 11.

Table 11

Stream	*Response*	*a*
S_1^k	2.01+0.2=1.21	5.99
S_2^k	2.01+0.2=2.21	6.99
S_3^k	3.00+0.2=3.20	0.00
S_4^k	4.00+0.2=4.20	0.00

Therefore, the message stream set is EDF schedulable, since $R_i^k \leq T_i^k$ (D_i^k), $\forall_i$, while it would not be schedulable by RM. In fact, stream S_4^k, and using equation (36), will have the following worst-case queuing delay:

$Q_4^{k(0)} = 1; Q_4^{k(1)} = 4; Q_4^{k(2)} = 5; Q_4^{k(3)} = 6; Q_4^{k(4)} = 7; Q_4^{k(5)} = 7$

and thus R_4^k = 7 + 0.2 = 7.2, which is larger than T_4^k (D_4^k)= 6.99. Fig. 7 puts this to evidence.

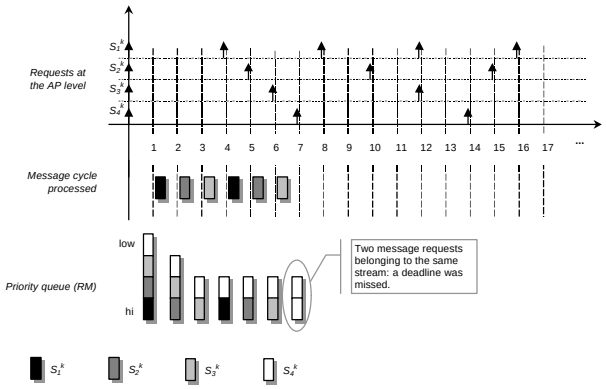

Figure 7

As a final remark, it is important to note that the stream set of Table 8 does not emphasise the importance of parameter *a*. This is only due to the specific characteristics of the particular stream set. In fact, the results in Tables 10 and 11 show that considering $a = 0$ corresponds virtually to the actual worst-case response time. The following example will better illustrate the importance of parameter *a* in the evaluation of the queuing delay. The only difference to the previous example is in the value of D_2^k.

Table 12

Stream	C_i^k	T_i^k	D_i^k
S_1^k	0.2	3.99	3.99
S_2^k	0.2	4.99	3.90
S_3^k	0.2	5.99	5.99
S_4^k	0.2	6.99	6.99

For this stream set example, the set of values for *a* would be ($L = 9$):

Table 13

Stream	$a = 0$	*a1*	*a2*	*a3*	*a4*	*a5*	*a6*	*a7*
S_1^k	0.00	2.00	3.00	3.99	4.90	7.98	7.99	---
S_2^k	0.00	0.09	2.09	3.09	4.08	4.99	8.07	8.08
S_3^k	0.00	1.00	1.99	2.90	5.98	5.99	7.89	7.99
S_4^k	0.00	0.99	1.90	4.98	4.99	6.89	6.99	8.97

Using the resulting values for each $Q_i^k(a)$, the difference ($Q_i^k(a) - a$) is:

Table 14

Stream	$a = 0$	*a1*	*a2*	*a3*	*a4*	*a5*	*a6*	*a7*
S_1^k	2.00	1.00	0.00	-0.99	1.10	-0.98	0.01	---
S_2^k	1.00	1.91	0.91	-0.09	-1.08	-1.99	-1.07	0.92
S_3^k	3.00	2.00	1.01	0.10	-2.98	-2.99	1.11	3.01
S_4^k	4.00	2.01	1.10	-1.98	-1.99	-3.89	-3.99	2.03

As it can be seen, for stream S_2^k, with $a = 0.09$, the queuing delay (as compared to the case of $a = 0$) increases from 1.00 to 1.91. This is an understandable result, as its "absolute deadline" will then be 0.09 + 3.90 = 3.99, and therefore, S_i^k will be scheduled earlier.

6. Conclusions

The main contribution of this paper was the adaptation, by providing the convenient analogies, of the feasibility tests available for non pre-emptive task scheduling to the scheduling of messages in SMTV token-passing networks.

We reasoned on how the blocking effect (resulting from non pre-emption) in the schedulability analysis of tasks could be mapped to each case of priority scheme used to schedule messages. We showed how the worst-case execution time of tasks could be translated to the upper bound of the token rotation time in SMTV token-passing networks. More important, we demonstrated how the simple utilisation-based feasibility tests for non pre-emptive independent tasks could be easily adapted to be used as (token) utilisation-based tests. However, as these tests can be quite pessimistic, we developed response-time tests which were also adapted from the well know response time tests used for RM/DM scheduled non pre-emptable independent tasks and we also adapted the more recently developed response time tests for EDF scheduled non pre-emptable independent tasks.

Acknowledgements

This work was partially supported by ISEP, FLAD, DEMEGI and FCT.

References

[1] Liu, C. and Layland, J. (1973). Scheduling Algorithms for Multiprograming in Hard-Real-Time Environment. In Journal of the ACM, Vol. 20, No. 1, pp. 46-61.

[2] Tindell, K. (1992). An Extendible Approach for Analysing Fixed Priority Hard Real-Time Tasks. Department of Computer Science, University of York, Technical Report YCS-189.

[3] Lehoczky, J. (1990). Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines. In Proceedings of the 11th IEEE Real-Time Systems Symposium, pp. 201-209.

[4] Sha, L., Rajkumar, R. and J. Lehoczky (1990). Priority Inheritance Protocols: an Approach to Real-Time Synchronisation. In IEEE Transactions on Computers, Vol. 39, No. 9, pp. 1175-1185.

[5] Joseph, M. and Pandya, P. (1986). Finding Response Times in a Real-Time System. In The Computer Journal, Vol. 29, No. 5, pp. 390-395.

[6] Audsley, N., Burns, A., Richardson, M., Tindell, K and Wellings, A. (1993). Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. In Software Engineering Journal, Vol. 8, No. 5, pp. 285-292.

[7] Baker, T. (1991). Stack-Based Scheduling of Real-Time Processes. In Real-Time Systems, Vol. 3, No. 1, pp. 67-99.

[8] Zheng, Q. (1993). Real-Time Fault-Tolerant Communication in Computer Networks. PhD Thesis, University of Michigan.

[9] Baruah, S., Howell, R., Rosier, L. (1990). Algorithms and Complexity Concerning the Pre-emptive Scheduling of Periodic Real-time Tasks on One Processor. In Real-Time Systems, 2, pp. 301-324.

[10] Ripoll, I., Crespo, A., Mok, A. (1996). Improvement in Feasibility Testing for Real-time Systems. In Real-Time Systems, 11, pp. 19-39.

[11] Spuri, M. (1995). Earliest Deadline Scheduling in Real-time Systems. PhD Thesis, Scuola Superiore Santa Anna, Pisa.

[12] Zheng, Q., Shin, K. (1994). On the Ability of Establishing Real-Time Channels in Point-to-Point Packet-Switched Networks. In IEEE Transactions on Communications, Vol. 42, No. 2/3/4, pp. 1096-1105.

[13] George, L., Rivierre, N., Spuri, M. (1996). Preemptive and Non-Preemptive Real-Time Uni-Processor Scheduling. Technical Report No. 2966, INRIA.

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