From Task Scheduling in Single Processor Environments to Message Scheduling in a PROFIBUS Fieldbus Network

Eduardo Tovar[1], Francisco Vasques[2]

[1] Department of Computer Engineering, Polytechnic Institute of Porto, Rua de São Tomé, 4200 Porto, Portugal
emt@dei.isep.ipp.pt
[2] Department of Mechanical Engineering, University of Porto, Rua dos Bragas, 4099 Porto Codex, Portugal
vasques@fe.up.pt

Abstract. In this paper we survey the most relevant results for the priority--based schedulability analysis of real-time tasks, both for the fixed and dynamic priority assignment schemes. We give emphasis to the worst-case response time analysis in non-preemptive contexts, which is fundamental for the communication schedulability analysis. We define an architecture to support priority-based scheduling of messages at the application process level of a specific fieldbus communication network, the PROFIBUS. The proposed architecture improves the worst-case messages' response time, overcoming the limitation of the first-come-first-served (FCFS) PROFIBUS queue implementations.

1 Introduction

Real-time computing systems are defined as those systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced [1]. There are various examples of real-time computing systems, ranging from distributed computer control to robotics. In this paper we particularly address distributed computer-controlled systems (DCCS) applications.

A recent trend in DCCS is to interconnect distributed elements by means of a multi-point broadcast network, instead of using traditional point-to-point links. As the network bus is shared between a number of network nodes, there is an access contention, which must be solved by the Medium Access Control (MAC) protocol.

Usually, a DCCS application imposes real-time constraints. In essence, by real-time constraints we mean that 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 MAC protocol is able to schedule messages streams according to its real-time requirements.

During the past decade a reasonable number of commercial solutions, usually called fieldbus networks, have been proposed to support DCCS applications. Some

distinguished examples are FIP [2], PROFIBUS [3], CAN [4] or P-NET [5]. In parallel, several international standardisation efforts have been and are still being carried out. One of the most relevant resulted into the European Standard EN 50170 [6], which basically encompasses three well-proven fieldbus national standards: PROFIBUS, FIP and P-NET.

A potential leap towards the use of fieldbus networks to support DCCS applications lies in the evaluation of its temporal behaviour. Several studies have been performed, such as those on CAN [7,8], on FIP [9,10] and on P-NET [11,12].

Contrarily to other networks [15-16] which are based in the timed token protocol [17], in PROFIBUS it is not possible to define, in each master, its synchronous bandwidth allocation[1]. Thus, it is not possible to use analysis similar to those proposed in [18-19]. In [13] the authors take two different approaches to guarantee real-time traffic using PROFIBUS networks. One of the approaches considers a worst-case scenario with, at most, one high priority message cycle[2] transmission per token visit. Thus, if there are *m* high priority messages pending to access the bus, in the worst-case it will take *m* token visits to execute all those high priority message cycles. Considering the maximum token cycle (has derived in [14]), it is possible to evaluate the maximum queuing delay of any message request.

In PROFIBUS, pending requests are queued in a First-Come-First-Served (FCFS) queue. Hence, a message request will have a worst-case queuing delay that depends on the maximum priority inversion of messages. This motivated a study on adding local priority-based scheduling mechanisms to PROFIBUS masters. One possibility relies on the implementation of a priority-based queue at the application process level, and limits the FCFS communication stack queue to one pending request. There are several results available for the pre-run-time schedulability analysis of tasks in a single processor environment. In this paper we will show how those results can be adapted for message pre-run-time schedulability analysis in PROFIBUS communication networks.

The remaining of the paper is organised as follows. In the next section we will survey the most relevant results of tasks' pre-run-time schedulability analysis for both fixed and dynamic priority assignment schemes. We will give emphasis to the worst-case response time analysis in non-preemptive contexts. In section 3, PROFIBUS protocol is presented and the timing analysis proposed in [13] is detailed. Finally, in section 4, an architecture to support priority-based scheduling of messages at the PROFIBUS application process level is defined, and the single processor worst-case tasks' response time analysis is adapted to obtain the worst-case messages' response time.

---

[1] In PROFIBUS there is no synchronous bandwidth allocation (H_i). If a master receives a late token only one high priority message may be transmitted. Contrarily, in the timed token protocol the station can transmit real-time (high-priority) traffic during H_i, even if the token is late.

[2] In PROFIBUS a message cycle consists on a master's request frame and responder's immediate response frame.

2 Schedulability Analysis of Real-Time Tasks

One of the most used priority assignment schemes is to give tasks a priority level based on its period: the shorter the period, the higher the priority. This assignment is intuitively explained by the fact that more critical devices will provide inputs more frequently (asynchronous interrupts), or indeed will be polled more frequently. Thus, if they have shorter periods, the worst-case response time should also be shorter. This type of priority assignment is known as the rate monotonic (RM) priority assignment.

If some of the tasks are sporadic, it may not be reasonable to consider its relative deadline equal to the period. A different priority assignment can be to give the tasks a priority level based on its deadline: the shorter the relative deadline, the higher the priority. This type of priority assignment is known as the deadline monotonic (DM) priority assignment [20].

In both RM and DM priority assignments, priorities are fixed, in the sense that they do not vary along time. At run-time, tasks are dispatched highest-priority-first. A similar dispatching policy can be used if the task that is chosen to run, is the one with the earliest deadline. This corresponds to a priority-driven scheduling where priorities of the tasks vary along time, hence corresponding to a dynamic priority assignment. This type of dynamic priority assignment is known as earliest deadline first (EDF) priority assignment [21].

In all three cases, the dispatching will take place when either a new task is released or the execution of the running task ends. This however may not stand in a non pre--emptive context. With priority-based scheduling, a higher-priority task may be released during the execution of a lower priority one. In a pre-emptive system, the higher-priority task will pre-empt the lower-priority one. Contrarily, in non-preemptive systems, the lower-priority task will be allowed to complete its execution before the other executes.

For the remaining of this paper, we characterise a task set (or a message stream set) by its maximum execution time (transmission time), its relative deadline and its period, denoted, respectively, as C_i, D_i and T_i.

2.1 Analysis for the Fixed-Priority Assignment

For the RM priority assignment, Liu and Layland [21] derived an utilisation-based pre-run-time schedulability test, which, if satisfied, guarantees that all n tasks will meet their deadlines: $\sum_{i=1,..,n} C_i/T_i < n \times (2^{1/n} - 1)$[3]. This utilisation-based test is valid for periodic independent tasks with relative deadlines equal to the period and for pre-emptive systems. For the non pre-emptive case, a similar analysis can be adapted from results available in [22]. The response time tests (by opposition to the utilisation-based tests) are more advantageous for pre-run-time schedulability analysis, since individual results can be obtained for each task.

---

[3] C is worst-case computation time of the task an T is the minimum time between task releases (period). If the task is sporadic, the minimum period between any two aperiodic releases is considered.

Joseph and Pandaya [23] proved that the worst-case response time r_i of a task t_i is found when all tasks are synchronously released at their maximum rate (critical instant). r_i is computed by the following recursive equation (where *hp(i)* denotes the set of tasks of higher priority than t_i): $r_i^{m+1} = C_i + \sum_{j \in hp(i)} \left(\lceil r_i^m / T_j \rceil \times C_j \right)$. The recursion ends when $r_i^{m+1} = r_i^m = r_i$ and can be solved by successive iterations starting from $r_i^0 = C_i$. Indeed, it is easy to show that r_i^m is non-decreasing. Consequently, the series either converges or exceeds T_i (in the case of RM) or D_i (in the case of DM). If the series exceeds T_i (D_i), the task t_i is not schedulable.

This result is valid for the preemptive context. Fewer results are known about fixed priority-based non-preemptive scheduling. In [24] Audsley *et al.* updated the analysis of Joseph and Pandaya [23] to include blocking factors introduced by periods of non-preemption. The worst-case response time is updated to:

$$r_i = w_i + C_i \tag{1}$$

where w_i is given by $w_i^{m+1} = B_i + \sum_{j \in hp(i)} \left(\lceil w_i^m / T_j \rceil \times C_j \right)$. B_i is the blocking factor of task t_i, that is, an upper bound on the time a lower priority (thus resulting in priority inversion) task can execute and prevent the execution of task t_i. A methodology for deriving B_i was introduced in [22] to solve the problem of non-independence of tasks, which can also be used for the schedulability analysis of non pre-emptive tasks:

$$B_i = \max_{\forall j \in lp(i)} \{ C_j \} \tag{2}$$

where *lp(i)* denotes the tasks with lower priority than *i*.

2.2 Dynamic-Priority Assignment

The EDF schedulability test was introduced in [21], for periodic tasks with relative deadlines equal to task periods. The main result from that work is the following utilisation-based test: $\sum_{i=1,\ldots,n} C_i/T_i < 1$. This result is hardly useful for real-time systems with sporadic tasks. If we consider the more general case of $D_i \leq T_i$ [25], the inequality can be updated to: $\forall_{t \geq 0}$, $\sum_{i=1,\ldots,n} \left(\lceil (t - D_i)/T_i \rceil^+ \times C_i \right) \leq t$, where $\lceil x \rceil^+ = 0$ if $x < 0$.

The proof for this inequality is intuitive. Assume that at time $t = 0$, there are no pending tasks. Then, a necessary condition to guarantee the tasks deadlines is that the amount of time, T, needed to transmit all tasks generated during [0, t] with absolute deadlines $\leq t$ is not greater than t. Since the minimal inter-arrival time for a task t_i is T_i, there are at most $\lceil (t - D_i)/T_i \rceil^+$ requests for that task during [0, t] which have a deadline $\leq t$. Those requests will need, at most, $\lceil (t - D_i)/T_i \rceil^+ \times C_i$ time to be completed. Thus the maximum value for T is given by $\sum_{i=1,\ldots,n} \left(\lceil (t - D_i)/T_i \rceil^+ \times C_i \right)$.

Note that if $D_i = T_i$, inequality $\forall_{t \geq 0}$, $\sum_{i=1,\ldots,n} \left(\lceil (t - D_i)/T_i \rceil^+ \times C_i \right) \leq t$ is satisfied if the utilisation-based test, $\sum_{i=1,\ldots,n} C_i/T_i < 1$, is satisfied, since in this case $\lceil (t - T_i)/T_i \rceil^+ \leq t/T_i$. The inequality $\forall_{t \geq 0}$, $\sum_{i=1,\ldots,n} \left(\lceil (t - D_i)/T_i \rceil^+ \times C_i \right) \leq t$ has a practical problem, since it must be checked over an infinite length interval [0, ∞). But, if we look to its left-hand interval, we can easily understand that its value only changes at $k \times T_i + D_i$ steps.

Additionally, there exists a point t_{max}, such that $\sum_{i=1,..,n}\left(\lceil (t - D_i)/T_i \rceil^+ \times C_i\right) \leq t$ always hold for$\forall_{t \geq t_{max}}$ (under the condition that the total utilisation $\sum_{i=1,..,n} C_i/T_i < 1$). Hence, the following represents the feasibility test for EDF dispatched tasks in a single processor pre-emptive context:

$$\forall_{t\in S}, \sum_{i=1}^{n}\left\lceil \frac{t-D_i}{T_i}\right\rceil^+ \times C_i \leq t, \quad \text{with } S = \left(\bigcup_{i=1}^{n}\{D_i + k\times T_i, k \in \aleph\}\right) \cap [0, t_{max}) \tag{3}$$

The determination of t_{max} has been addressed in several works [26-29]. Note that when the utilisation approaches 1, t_{max} becomes very large. This is the main difficulty for such utilisation-based schedulability test.

For the non-preemptive case, a similar feasibility test was derived in [25,30]:

$$\forall_{t\geq D_{min}}, \sum_{i=1}^{n}\left\lceil \frac{t-D_i}{T_i}\right\rceil^+ \times C_i + \max_{i=1,...,n}\{C_i\} \leq t, \quad \text{with } D_{min} = \min_{i=1,...,n}\{D_i\} \tag{4}$$

In [31], George *et al.* argue about the pessimism of (4). The main argument is that in [30] Zheng and Shin consider that the cost of possible priority inversions, caused by task non pre-emptability, is always initiated by the longest task and, moreover, is effective during all the studied interval. To reduce the pessimism level of (4), they suggest the following update (the values for *S* are precisely defined in [31]):

$$\forall_{t\in S}, \sum_{i=1}^{n}\left\lceil \frac{t-D_i}{T_i}\right\rceil^+ \times C_i + \max_{\substack{i=1,...,n \\ D_i>t}}\{C_i - 1\} \leq t, \quad \text{with } \max_{\substack{i=1,...,n \\ D_i>t}}\{C_i - 1\} = 0 \text{ if } \nexists_i : D_i > t \tag{5}$$

The worst-case response time analysis for preemptive EDF scheduling was first studied in [32]. The starting basis for such analysis was that the worst-case response time for a general task set is not necessarily obtained with a synchronous pattern arrival (that is, considering the critical instant as defined for the fixed priority case). In that work, Spuri showed that the worst-case response time of a task t_i is found in the deadline busy period[4] of the processor.

This means that, in order to find the worst-case response time of t_i, we need to examine several scenarios within which, while t_i has an instance released at time *a*, all other tasks are synchronously released. If the start time of the *asap*[5] pattern of t_i is denoted by S_i then $S_i = 0$, $\forall_{j\neq i}$. In general, t_i may also have other instances released earlier than *a*. In particular, its start time is $S_i = a - \lfloor a/T_i \rfloor \times T_i$. Thus, given a value of *a*, the response time of the t_i 's instance released at time *a* is:

$$r_i(a) = \max\{C_i, L_i(a) - a\} \tag{6}$$

[4] The longest deadline busy period (the processor is fully utilised) appears when all tasks but task *i* are synchronously released and at their maximum rate.

[5] as soon as possible

where $L_i(a)$ is the length of the busy period. $L_i(a)$ can be evaluated by the following iterative computation (starting with $L_i^0=0$): $L_i^{m+1}(a) = W_i(a, L_i^m(a)) + \left(1+\lfloor a/T_i \rfloor\right)\times C_i$, with $W_i(a, t) = \sum_{j\neq i, \text{ and } Dj \leq a+Di} \left(\min\left\{\lceil t/T_j \rceil, 1+\lfloor (a+D_i-D_j)/T_j \rfloor\right\}\times C_j\right)$.

The worst-case response time for a task t_i is then given:

$$r_i = \max_{a\geq 0}\{r_i(a)\} \tag{7}$$

The remaining problem is for which values of a r_i must be evaluated? If we look to the right-hand side of the $W_i(a, t)$ equation, we can easily understand that its value does only changes at $k\times T_j+D_j-D_i$ steps. Thus,

$$a \in \bigcup_{j=1}^{n}\left\{k\times T_j + D_j - D_i, k \in \aleph\right\} \cap [0, L_i] \tag{8}$$

where L_i is the maximum length of the deadline busy periods, as defined in [32].

The worst-case response time analysis for the non pre-emptive EDF scheduling was introduced in [31]. The main difference to the analysis made for the pre-emptive case is that a task instance with a later absolute deadline can possibly cause a priority inversion. Thus, instead of analysing the busy period preceding its completion time, 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:

$$r_i(a) = \max\{C_i, L_i(a) + C_i - a\} \tag{9}$$

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 (also starting with $L_i^0 = 0$): $L_i^{m+1}(a) = \max_{Dj>a+Di}\{C_j - 1\} + W_i^*(a, L_i^m(a)) + \lfloor a/T_i \rfloor \times C_i$, with $W_i^*(a, t) = \sum_{j\neq i, \text{ and } Dj \leq a+Di} \left(\min\left\{1+\lfloor t/T_j \rfloor, 1+\lfloor (a+D_i-D_j)/T_j \rfloor\right\}\times C_j\right)$.

Again, a should be evaluated in the following set of values:

$$a \in \bigcup_{j=1}^{n}\left\{k\times T_j + D_j - D_i, k \in \aleph\right\} \cap [0, L] \tag{10}$$

where, in this case, L is the length of the sychrounous busy period, which can be evaluated by (starting with $L^0 = \sum_{i=1,..,n} C_i$): $L_i^{m+1} = W(L^m)$, with $W(t)=\sum_{i=1,..,n} \lceil t/T_i \rceil \times C_i$, which is solved by reoccurrence.

3. PROFIBUS Timing Analysis

3.1 PROFIBUS Timed Token Protocol

The PROFIBUS MAC protocol is based on a token passing procedure, used by master stations to grant the bus access to each one of them, and a master-slave procedure used by master stations to communicate with slave stations. One of the PROFIBUS

MAC main functions is the control of the token cycle time, which will now be briefly explained.

After receiving the token, the measurement of the token rotation time begins. This measurement expires at the next token arrival and results in the real token rotation time (T_{RR}). A target token rotation time (T_{TR}) must be defined in a PROFIBUS network. The value of this parameter is common to all masters, and is used as follows. When a station receives the token, the token holding time (T_{TH}) timer is given the value corresponding to the difference, if positive, between T_{TR} and T_{RR}. PROFIBUS defines two categories of messages: high priority and low priority. These two categories of messages use two independent outgoing queues. If at the arrival, the token is late, that is, the real token rotation time (T_{RR}) was greater than the target rotation time (T_{TR}), the master station may execute, at most, one high priority message cycle. Otherwise, the master station may execute high priority message cycles while $T_{TH} > 0$. T_{TH} is always tested at the beginning of the message cycle execution. This means that once a message cycle is started it is always completed, including any required retries, even if T_{TH} expires during the execution. We denote this occurrence as a T_{TH} overrun. The low priority message cycles are executed if there are no high priority messages pending, and while $T_{TH} > 0$ (also evaluated at the start of the message cycle execution, thus leading to a possible T_{TH} overrun).

Below is a description of the PROFIBUS token passing algorithm:

```
/* initialisation procedure */
At each station k, DO:
T_TH ← 0 ;
T_RR ← 0 ;
Start T_RR ;          /* count-up timer */

/* run-time procedure */
At each station k, at the Token arrival, DO:
T_TH ← T_TR - T_RR ;
T_RR ← 0 ;
Start T_RR ; /* count-up timer */
IF T_TH > 0 THEN
         Start T_TH  /* count-down timer */
ENDIF;
IF waiting High priority messages THEN:
           Execute one High priority message cycle
ENDIF;
WHILE T_TH > 0 AND pending High priority message cycles DO
Execute High priority message cycles
ENDWHILE;
WHILE T_TH > 0 AND pending Low priority message cycles DO
         Execute Low priority message cycles
ENDWHILE;
Pass the token to station (k + 1) (modulo n);
```

As mentioned in section 1, in PROFIBUS, a message cycle consists on a master's action frame (request or send/request frame) and the responder's immediate acknowledgement or response frame. User data may be transmitted in the action frame or in the response frame. Note that in PROFIBUS a master station is allowed to send up to a limited number of retries, if the response does not come within a

predefined time. Hence, the message cycle time length must also include the time needed to process the allowed retries.

3.2 Message Worst-Case Response Time

As a master station is able to transmit, at least, one high priority message per received token (no matter if there is enough token holding time left), a maximum queuing delay can be guaranteed for PROFIBUS messages. Defining T_{cycle} as the upper bound between two consecutive token arrivals to a particular master, the maximum queuing delay of a single message request (Q) is equal to T_{cycle}. Note that this only guarantees a maximum transmission delay for the first high priority message in the outgoing queue. If there are m pending messages in the outgoing queue it will take, in the worst-case, m token visits to execute all those high priority messages.
It is obvious that the queuing delay depends on the outgoing high priority queue implementation. PROFIBUS implements First-Come-First-Served (FCFS) outgoing queues. Consequently, if nh^k represents the number of high priority message streams[6] in a master k, then the maximum number of pending messages will be nh^k, corresponding to one message per each high priority message stream (Sh_i^k) (two messages from the same stream would mean that a deadline for that message stream was missed). Thus, an upper bound for the message queuing delay in a master k is: $Q_i^k = nh^k \times T_{cycle} - Ch_i^k$, where Ch_i^k denotes the maximum length (request, response, turnaround time and maximum allowable retries) of a message from stream Sh_i^k. The worst-case response time for a message cycle is given by:

$$R_i^k = Q_i^k + Ch_i^k = nh^k \times T_{cycle} \quad (11)$$

Thus, a set of PROFIBUS high priority message streams is guaranteed if the following pre-run-time schedulability condition is verified:

$$Dh_i^k \geq R_i^k, \ \forall_{master\ k,\ stream\ Sh_i^k} \quad (12)$$

that is, if relative deadlines of all high priority message streams are greater than or equal to their worst-case communication response time. If (12) is not satisfied, the high priority traffic is not schedulable. In order to solve inequality (12), we need to evaluate T_{cycle}.

3.3 Cycle Time Evaluation

As shown in [13,14], T_{cycle} can be expressed as a function of T_{TR}. Thus, having defined T_{cycle}, it is possible to set the T_{TR} parameter in order to satisfy the pre-run-time schedulability condition (12).

Considering that T_{cycle} is the upper bound of the real token rotation time (T_{RR}), we must reason about T_{RR} in order to evaluate the value of T_{cycle}. The T_{RR} value will be

[6] A message stream corresponds to a temporal sequence of message cycles related, for instance, with the reading of a process sensor or the updating of a process actuator.

smaller than T_{TR} (that is, the token will always be in advance to its schedule), except if one or more masters in the logical ring cause the token to be late. The main cause for the token lateness (T_{del}) is the T_{TH} overrun. Such token lateness may be worsened if the following masters, having received a late token, still transmit, each one, one high priority message.

Considering the worst-case scenario in which a master k overruns its T_{TH} and the following masters until master k–1 (modulo n) receive and use a late token, T_{del} can be evaluated as follows:

$$T_{del} = \sum_{k=1}^{n} \left(C_M^k \right) \quad (13)$$

where C_M^k stands for the longest message cycle associated to each master k: $C_M^k = \max\{\max_{i=1,..,nh^k}\{Ch_i^k\}, Cl^k\}$, and Cl^k is the longest low priority message cycle in a master k. Thus, T_{cycle} can be evaluated as follows:

$$T_{cycle} = T_{TR} + T_{del} \quad (14)$$

To illustrate the T_{cycle} evaluation, assume the following scenario: after a token cycle without message transmissions, master k receives the token ($T_{TH}^{(k)} = T_{TR} - \tau$, since $T_{RR}^{(k)} = \tau$)[7]. This master can actually hold the token during $T_{TH}^{(k)}$ plus the duration of its longest message. In this situation all the following masters in the logical ring will receive a late token, thus only being able to process, at most, one high priority message cycle.

A more accurate definition for T_{cycle} can be found in [14], where factors such as the different message cycle lengths and the relative position of each master in the logical ring are taken into consideration for the definition of T_{cycle}.

3.4 Setting the PROFIBUS T_{TR} Parameter

We can now update the pre-run-time schedulability condition: $Dh_i^k \geq nh^k \times (T_{TR} + T_{del})$. Therefore, the T_{TR} parameter can be set as follows:

$$0 \leq T_{TR} \leq \frac{Dh_i^k}{nh^k} - T_{del}^k, \forall_{\text{master } k,\ \text{stream } Sh_i^k} \quad (15)$$

4 Adding Priority-Based Message Scheduling

The first-come-first-served (FCFS) dispatching of the PROFIBUS outgoing queue may induce in a "priority inversion" with the length ns^k–1 as, in the worst-case, a message request with a more stringent deadline may be placed in the outgoing queue after the other ns^k–1message requests of the same master station.

[7] The parameter t includes the ring latency and other protocol and network overheads.

A priority-based queue would solve this problem: more stringent messages would have lower worst-case response times whereas less stringent messages would have higher worst-case response times (as compared to the FCFS dispatching policy).

Without changing the PROFIBUS FCFS implementation, a priority-based queue may be implemented at the application process level, provided that the communication stack outgoing queue (FCFS based) is limited to one pending request. In PROFIBUS, this length control of the communication stack outgoing queue can be trivially achieved by the proper use of a local management service.

4.1 Message's Release Jitter

In the analysis we made in section 3, the periods of the messages were not relevant for the evaluation of the worst-case message's response time. Independently of the model of the tasks that generate the message requests, the worst-case queuing delay occurs when ns^k requests are placed at the outgoing queue just before master k passes the token to the subsequent station.

If, in each master, messages are dispatched using a priority-based scheduling policy (either DM or EDF can be considered), then, by using similar worst-case response time analysis as for the case of non pre-emptive[8] task scheduling in single processor environments, periods of the message request are now relevant. In fact, both equation (1) (for the case of pre-run-time schedulability analysis of non pre-emptive DM) and equation (9) (for the case of pre-run-time schedulability analysis of non pre--emptive EDF) very much depend on the periods of the tasks (now message requests).

This rises the problem of message release jitter, which in the context of communication networks has been addressed in several studies such as by Tindell and Clark [33] and by Spuri [34].

In the case of PROFIBUS we can assume that message requests are placed in the priority-ordered application process queue by an application task, and a task's instance will generate a message stream request. In this sense we can say that messages inherit from sending tasks both their period and priority level (if fixed priorities are used). It is implicit that tasks at the application process level are scheduled according to a priority-based policy, most probably in a preemptive context. It results from this inheritance approach that the message requests generated by tasks will have a minimum inter-arrival time smaller than the period of the tasks, which generate them.

In PROFIBUS, the sending task and the response's receiving task are in the same host processor. In fact, they can even be the same task. There is an initial part of the task responsible for placing the request in the priority-based AP queue (during this process, this task competes with the other tasks for the processor). Then the task auto-suspends itself until the response arrives. Finally, after receiving the response, it processes it the task until completion (again in this phase competing with other running tasks). If this is the case, the messages's release jitter is the worst-case

[8] The pre-emptive schedulability analysis is not useful for message scheduling in PROFIBUS networks, as a message cycle is non pre-emptable.

response time of the first part of the task (which includes placing the request in the priority ordered queue).

We can think also about a model where sending and receiving tasks are separate tasks (in fact each pair of sending/receiving tasks do not correspond to independent tasks, since if one is runnable the other is not in the runnable queue). In this case, the release jitter for a message request corresponds to the worst-case response time of the sending task. That is, in a particular instance of the task, the message can be released close to the worst-case response time of the task; and in the subsequent release of the task, the message can be released as soon as the arrival of that new task's instance.

Independently of the task model adopted for the application level, we denote the release jitter of a message stream i in a master k as J_i^k. The evaluation of its value depends on the adopted task model and also on the scheduling policy of the tasks. Both task models demand some careful for analysing tasks' worst-case response times.

4.2 End-To-End Communication Delay

The inclusion of an application process task model motivates the definition of the end-to-end communication delay [35]. With this concept, we associate timing requirements to tasks, and messages inherit periods, priorities and release jitter from tasks. If we use EDF scheduling, messages will be placed in the priority-based queue according to the earliness of the absolute deadline of the message's generating task. Note that the AP priority-based queue should only be re-ordered when a new request is generated (this stands for both DM and EDF).

The end-to-end communication delay (E) can be generically defined as: $E=g+Q+C+d$. g represents the worst-case generation delay for the master application task to generate and queue a specific message request. This would also corresponds to the message release jitter, which will be used for computing Q. The term Q corresponds to the worst-case delay for that request to gain access to the communications device after being queued. As previously mentioned, using a priority-based dispatching, this parameter would depend on the release jitter of each message stream. The term C corresponds not only to the worst-case for transmitting the request, but also to the time needed to receive the response from the slave (including processing at the slave side, slave turnaround time, propagation delay and other network latencies). Finally, d represents the delivery delay, that is, the time needed to process the response before finally delivering it to the destination task, which in the case of PROFIBUS is in the same host processor as the sending task. Variation to this results from whether separated sending and receiving tasks are considered.

In the next sub-section we will simply focus on the Q (or $Q+C$) parameter, and redefine the worst-case message's response time. Basically we are going to update equation (11) for embodying a priority-based dispatching policy.

4.3 Priority-Based Message's Response Time

Basically, the *Cs* in equations (1) and (9) will be replaced by T_{cycle}, and for considering one "blocking" the requests can appear marginally after receiving the token and marginally before passing the token. Assume also that all message cycles are equal. Hence, considering DM, equation (11) can be updated to:

$$R_i^k = T_{cycle}^* + \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{R_i^k + J_j^k}{T_j^k} \right\rceil \times T_{cycle} \right) \quad (16)$$

where $T^*_{cycle}=T_{cycle}$ except for the case of the message with the lowest priority (in this case $T^*_{cycle}=0$).

Considering EDF, equation (11) can be updated to:

$$R_i^k(a) = \max\left\{T_{cycle}, L_i(a) + T_{cycle} - a\right\} \quad (17)$$

where (starting with $L_i^0(a)=0$):

$$\begin{aligned} L_i^{m+1}(a) &= \max_{D_j^k > a + D_i^k} \left\{T_{cycle} - 1\right\} + W_i^*\left(a, L_i^m(a)\right) + \left\lfloor \frac{a}{T_i^k} \right\rfloor \times T_{cycle}, \text{ with} \\ W_i^*(a,t) &= \sum_{\substack{j \neq i \\ D_j^k \leq a + D_i^k}} \left(\min\left\{1 + \left\lfloor \frac{t + J_j^k}{T_j^k} \right\rfloor, 1 + \left\lfloor \frac{a + D_i - D_j + J_j^k}{T_j^k} \right\rfloor \right\} \times T_{cycle} \right) \end{aligned} \quad (18)$$

5 Conclusions

In this paper we have drawn a comprehensive study on how to use PROFIBUS to support real-time communications. We surveyed previous relevant work on how to evaluate the worst-case response time of tasks in a single processor environment. Particular relevance was devoted to the non pre-emptive case.

As in PROFIBUS the message requests are processed in a first-come-first-served basis, we reasoned about the possibility of supporting priority-based scheduling of message requests at the application process level of PROFIBUS.

Finally, we have derived the worst-case response time of PROFIBUS message requests for both the cases of message requests dispatched using a DM priority assignment and an EDF priority assignment.

The relevance of this work results from the obvious conclusion that the use of priority-based dispatching mechanism at the application process level allows the support of messages with more tight deadlines.

Acknowledgements

This work was partially supported by FLAD under the project SISTER 471/97 and by ISEP under the project REMETER.

References

1. Stankovic, J.: “Real-Time Computing Systems: the Next Generation”, *in* STANKOVIC J., RAMAMRITHAM, K. (Eds.) “Tutorial: Hard Real-Time Systems” (IEEE, 1988), pp. 14-38, 1988.
2. Normes FIP NF C46-601 to NF C46-607, Union Technique de l'Electricité, AFNOR, 1990.
3. Profibus Standard DIN 19245 part I and II. Translated from German, Profibus Nutzerorganisation e.V., 1992.
4. SAE J1583, Controller Area Network (CAN), an In-Vehicle Serial Communication Protocol. SAE Handbook, Vol. II, 1992.
5. The P-NET Standard. International P-NET User Organisation ApS, 1994.
6. General Purpose Field Communication System, Vol. 1/3 (P-NET), Vol. 2/3 (Profibus), Vol. 3/3 (FIP), CENELEC, 1996.
7. Tindell, K., Hansson, H., Wellings, A.: “Analysing Real-Time Communications: Controller Area Network (CAN)”. Proceedings of the IEEE Real Time Systems Symposium (RTSS’94), S.Juan, Puerto Rico, pp. 259-263, IEEE Press, 1994.
8. Tindell, K., Burns, A., Wellings, A.: “Calculating Controller Area Network (CAN) Message Response Times”. Control Engineering Practice, Vol. 3, No. 8, pp. 1163-1169, Pergamon, 1995.
9. Raja, P., Ruiz, L., Decotignie, J.-D.: “On the Necessary Real-Time Conditions for the Producer-Distributer-Consumer Model”. Proceedings of 1st IEEE Workshop on Factory Communication Systems (WFCS'95), Leysin, Switzerland, 1995.
10. Pedro, P., Burns, A.: “Worst Case Response Time Analysis of Hard Real-Time Sporadic Traffic in FIP Networks”. Proceedings of 9th Euromicro Workshop on Real-time Systems, Toledo, Spain, pp. 5-12, 1997.
11. Tovar, E., Vasques, F.: “Pre-run-time Schedulability Analysis of P-NET Networks”. Proceedings of 24th Annual Conference of the IEEE Industrial Electronics Society (IECON'98), Aachen, Germany, pp. 236-241, 1998.
12. Tovar, E., Vasques, F., Burns, A.: “Real-Time Communications in Multihop P-NET Networks”. Submitted to Control Engineering Practice, 1998.
13. Tovar, E., Vasques, F.: “Real-Time Fieldbus Communications Using Profibus Networks”. To appear in the IEEE Transactions on Industrial Electronics, 1998.
14. TOVAR, E., VASQUES, F.: “Cycle Time Properties of the Profibus Timed Token Protocol”, submitted to IEE Proceedings - Software, 1998.
15. ISO, Information Processing Systems - Fibre Distributed Data Interface (FDDI) - Part 2: Token Ring Media Access Control (MAC), ISO International Standard 9314-2, 1989.
16. IEEE, IEEE Standard 802.4: token passing bus access method and physical layer specification, 1985.
17. Grow, R.: “A Timed Token Protocol for Local Area Networks”. Proceedings of Electro’82, May 1982, Token Access Protocols, Paper 17/3.
18. Agrawal, G., Chen, B., Zhao, W., Davari, S.: “Guaranteeing Synchronous Message Deadlines with the Timed Token Protocol”. Proceedings of the 12th IEEE International Conference on Distributed Computing Systems, June 1992.

19. Montuschi, P., Ciminiera, L., Valenzano, A.: “Time Characteristics of IEE802.4 Token Bus Protocol”. IEE Proceedings, January 1992, 139 (1), pp. 81-87.
20. Burns, A.: “Scheduling Hard Real-Time Systems”. Software Engineering Journal - Special Issue on Real-time Systems, pp. 116-128, May 1991.
21. Liu, C., Layland, J.: “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment”. Journal of the Association for Computing Machinery (ACM), Vol. 20, NO. 1, pp. 46-61, January 1973.
22. Sha, L., Rajkumar, R., Lehoczky, J.: “Priority Inheritance Protocols: an Approach to Real-Time Synchronisation”. IEEE Transactions on Computers, Vol. 39, NO. 9, pp. 1175-1185, September 1990.
23. Joseph, M., Pandya, P.: “Finding Response Times in a Real-Time System”. The Computer Journal, Vol. 29, NO. 5, pp. 390-395, 1986.
24. Audsley, N., Burns, A., Richardson, M., Tindell, K, Wellings, A.: “Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling”. Software Engineering Journal, Vol. 8, NO. 5, pp. 285-292, September 1993.
25. Zheng, Q.: “Real-Time Fault-Tolerant Communication in Computer Networks”. PhD Thesis, University of Michigan, 1993.
26. Baruah, S., Howell, R., Rosier, L.: “Algorithms and Complexity Concerning the Pre-emptive Scheduling of Periodic Real-time Tasks on One Processor”. Real-Time Systems, 2, pp. 301-324, 1990.
27. Baruah, S., Mok, A., Rosier, L.: “Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor”. Proceedings of the 11th Real-Time Systems Symposium (RTSS’90), pp. 182-190, 1990.
28. Ripoll, I., Crespo, A., Mok, A.: “Improvement in Feasibility Testing for Real-time Systems”. Real-Time Systems, 11, pp. 19-39, 1996.
29. Spuri, M.: “Earliest deadline Scheduling in Real-time Systems”. PhD Thesis, Scuola Superiore Santa Anna, Pisa, 1995.
30. Zheng, Q., Shin, K.: “On the Ability of Establishing Real-Time Channels in Point-to-Point Packet-Switched Networks”. IEEE Transactions on Communications, Vol. 42, no. 2/3/4, pp. 1096-1105, 1994.
31. George, L., Rivierre, N., Spuri, M.: “Preemptive and Non-Preemptive Real-Time Uni-Processor Scheduling”. Technical Report No. 2966, INRIA, September 1996.
32. Spuri, M.: “Analysis of Deadline Scheduled Real-Time Systems”. Technical Report No. 2772, INRIA, January 1996.
33. Tindell, K., Clark, J.: “Holistic Schedulability Analysis for Distributed Hard Real-Time Systems”, in Microprocessors and Microprogramming, No. 40, 1994.
34. Spuri, M.: “Holistic Analysis for Deadline Scheduled Real-Time Distributed Systems”. INRIA, Technical Report no. 2873, April 1996.
35. Tindell, K., Burns, A., Wellings, A.: “Analysis of Hard Real-Time Communications”. Real-Time Systems, 1995, 9, pp. 147-171.