Adding Local Priority-Based Dispatching Mechanisms to P-NET Networks: a Fixed Priority Approach

Eduardo Tovar
Dept. of Computer Engineering,
Polytechnic Institute of Porto,
Portugal
e-mail: emt@dei.isep.ipp.pt

Francisco Vasques
Dept. of Mechanical Engineering,
University of Porto,
Portugal
e-mail: vasques@fe.up.pt

Alan Burns
Dept. of Computer Science,
University of York,
United Kingdom
e-mail: burns@cs.york.ac.uk

Abstract

In this paper we address the real-time capabilities of P-NET, which is a multi-master fieldbus standard based on a virtual token passing scheme. We show how P-NET's medium access control (MAC) protocol is able to guarantee a bounded access time to message requests. We then propose a model for implementing fixed priority-based dispatching mechanisms at each master's application level. In this way, we diminish the impact of the first-come-first-served (FCFS) policy that P-NET uses at the data link layer. The proposed model rises several issues well known within the real-time systems community: message release jitter; pre-run-time schedulability analysis in non pre-emptive contexts; non-independence of tasks at the application level. We identify these issues in the proposed model and show how results available for priority-based task dispatching can be adapted to encompass priority-based message dispatching in P-NET networks.

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]. Distributed real-time systems are those systems that have timeliness requirements and include several inter-operating computing components.

In distributed real-time systems, the co-operation between the different computational devices is supported by some kind of inter-processor communication. Back-plane buses, like Futurebus+ [2] or VME [3] are an interesting solution when processors are in the same physical platform, or indeed very close to each other. Instead, if the processors are physically distant, real-time serial communication networks must be used.

Two main application areas have been emerging for real-time communication networks. One is the multimedia domain, which includes digital audio and video transmission. Another concerns the factory automation and process control systems. In the former, humans interact with the communication services and, in one way or another, set the real-time requirements. In the latter, however, communication needs are much more a system design issue, and thus, is the physical environment (process or mechanical apparatus which is being controlled) that imposes the real-time requirements. In this paper we address real-time communications for distributed computer controlled systems (DCCS), which are included in the factory automation and process control systems.

The communication requirements of DCCS triggered the need for specific serial communication networks [4,5]. Consequently, during the past decade or so, a significant number of industrial communication networks, usually called fieldbus networks, have been proposed to support DCCS applications. Some distinguished examples are FIP [6], PROFIBUS [7], CAN [8] and P-NET [9]. 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 [10], which encompasses three different fieldbus profiles: FIP, PROFIBUS and P-NET.

In multi-point broadcast networks, the network bus is shared between a number of nodes, which means that there is access contention. This access contention is arbitrated by the medium access control (MAC) protocol. To guarantee a bounded access time, the MAC protocol must be deterministic, meaning that the bus access delay must be bounded. In FIP, the determinism is guaranteed by a bus arbitrator, which controls periodic data transfers according to a static scanning table. The real-time capabilities of FIP have been widely studied, and [11,12] are just some examples. PROFIBUS adopts a simplified version of the timed token (TT) protocol [13]. Despite some differences to the TT protocol used in FDDI or IEEE802.4, for which real-time characterisation have been extensively addressed ([14,15] are just some examples), it is still possible to guarantee real-time behaviour with PROFIBUS networks [16,17]. CAN is itself a priority bus, which adopts a collision avoidance version of the well-known CSMA/CD protocol. In [18,19] the authors show how to guarantee real-time behaviour with CAN networks.

In [20] the authors analyse the P-NET's MAC behaviour and propose a worst-case response time (WCRT) analysis for P-NET messages. Such analysis on the knowledge of the maximum virtual token rotation time. Later, this analysis is improved in [21] by considering the actual token utilisation, instead of considering always the worst-case token rotation time. Both studies show however that the WCRT is very much penalised by the first-come-first-served (FCFS) policy adopted by P-NET for processing the pending requests. This motivated us to propose a model for adding local priority-based dispatching mechanisms to P-NET masters. Such dispatching mechanisms must be performed at the application process level, to preserve the compatibility with the P-NET standard. Figure 1 depicts the proposed architecture. A priority ordered queue is implemented at each master's application level and the FCFS communication stack outgoing queue is limited to one pending request.

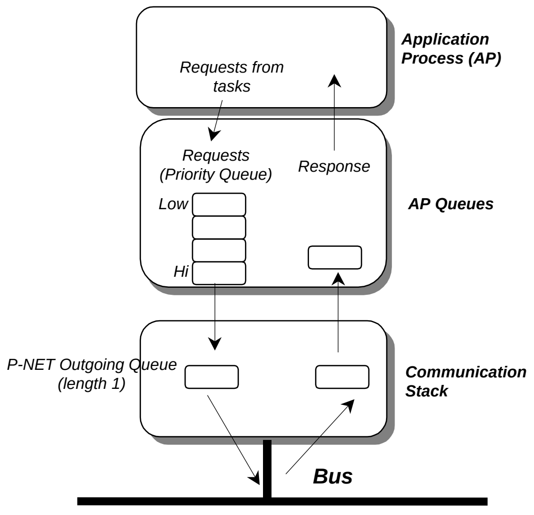

Figure 1. Proposed Architecture.

Each message request generated at the application level is placed in a priority ordered queue. Each time the communication stack queue becomes empty, a pending request may be transferred from the priority ordered queue.

Different schemes for priority assignment could be considered. We opt for fixed priority with priorities assigned by the deadline monotonic (DM) scheme.

The remaining of this paper is organised as follows. In section 2 we describe the concepts behind the P-NET's MAC mechanisms. In section 3 we give the worst-case communication response time analysis for P-NET's message requests, which were first developed in [20,21]. In section 4 we introduce the proposed model for adding local priority-based dispatching mechanisms to P-NET masters. The proposed model rises several issues well known within the real-time systems community: release jitter; pre-run-time schedulability analysis in non pre-emptive contexts; non-independence of tasks at the application process level. We identify these issues in the proposed model and then survey the most significant results available from the field of single processor task scheduling. We will adapt those results for the case of message scheduling and update the analysis made in section 3 to consider DM priority-based message dispatching. Finally, in section 5, we give conclusions.

2. P-NET's Virtual Token Passing Protocol

P-NET is a multi-master protocol, where a master sends a request and the addressed slave immediately returns a response. For multi-master support, P-NET uses a Virtual Token Passing (VTP) scheme.

The VTP scheme is implemented using two protocol counters. The first one, the Access Counter (AC), holds the node address of the currently transmitting master. When a request has been completed and the bus has been idle for 40 bit periods (520μs @ 76,8Kbps[1]), each one of the ACs is incremented by one. The master whose AC value equals its own unique node address is said to hold the token, and is allowed to access the bus. When the AC is incremented as it exceeds the "maximum No of Masters", the AC in each master is pre-set to one. This allows the first master in the cycling chain to gain access again. The second counter, the Idle Bus Bit Period Counter (IBBPC), increments for each inactive bus bit period. Should any transactions occur, the counter is re-set to zero. As explained above, when the bus has been idle for 40 bit periods following a transfer, all ACs are incremented by one, and the next master is thus allowed to access the bus. If a master have nothing to transmit (or indeed is not even present), the bus will continue inactive. Following a further period of 130μs (10 bit periods), the IBBPC will have reached 50, (60, 70,…) all the ACs will again be incremented, allowing the next master access. The virtual token passing will continue every 130μs, until a master does require access.

[1] The P-NET standard uses a data rate of 76800 bps. This data rate resulted from weighing up the conflicting requirement for data to be transported as fast as possible, but not at such speed as to negate the use of standard microprocessor UARTS, or restrict the usable distance or cable type [22].

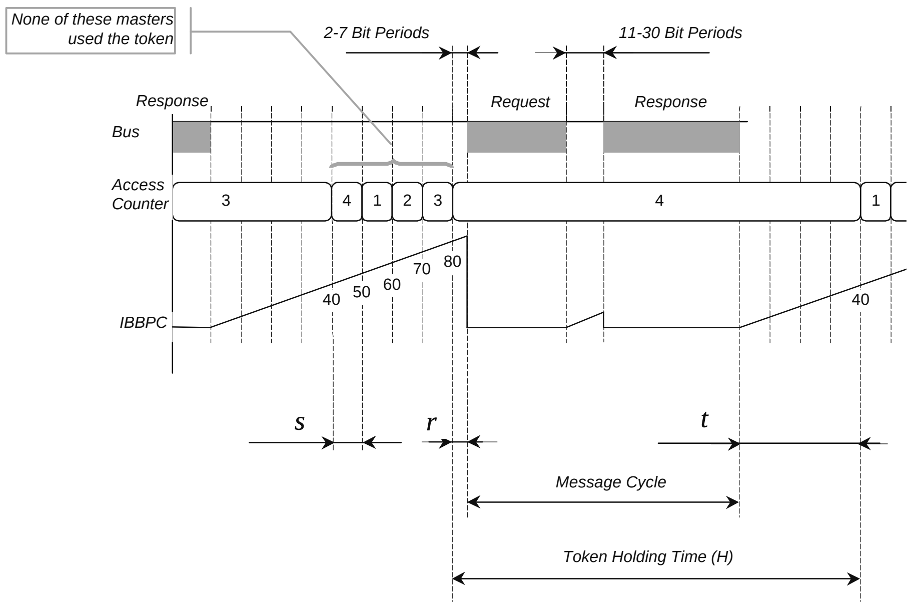

Figure 2. VTP Timing Example.

The P-NET standard also stands that each master is allowed to perform only one message transaction (later on defined as message cycle) per token visit. This is an important notion for the remaining of the paper.

After receiving the token, the master must transmit a request before a certain time has elapsed. This is denoted as the master's reaction time, and the standard imposes a worst-case value up to 7 bit periods). A slave is allowed to access the bus, between 11 and 30 bit periods after receiving a request, measured from the beginning of the stop bit in the last byte of the frame. The maximum allowed delay is then 390μs (corresponding to 30 bit periods). Later on, this delay will also be denoted as the slave's turnaround time.

As already stressed, at each token visit, a master may perform at most one message cycle. A message cycle consists on a master's request followed by the addressed slave's response. Assume that C_M is the maximum transmission duration of a message cycle in a P-NET network.

We define the maximum token holding time as:

$$H = r + C_M + t \tag{1}$$

In equation (1), t (= 40 bit periods) corresponds to the time to pass the token after a message cycle has been performed. r ($\leq$ 7 bit periods) denotes the worst-case master's reaction time. If a station does not use the token to perform a message cycle, the bus will be idle during s (= 10 bit periods) before all ACs are incremented. For better understanding both the basic MAC procedures and this notation, refer to figure 2.

Note that s (= 10 *bp*) is normally much shorter than H, since H includes both the request and response frames. In fact, a P-NET frame is limited (if network segmentation is not supported) to 69 bytes. As each frame byte actually corresponds to 11 bits[2], a frame may have up to 759 bits (69 x 11 bits). Thus, considering the case that both the request and response frames have 759 bits (more realistically either the request will be longer - case of writing data to a slave - or the response will be longer - case of receiving data from a slave), the overall sum for the token holding (H) time may go up to 1595 bit periods, corresponding to 20.8 ms @ 76800 bps.

3. Worst-Case Response Time Analysis

3.1. Network and Message Models

We consider a network with n masters, with addresses ranging from 1 to n. Each master accesses the network according to the VTP scheme. Hence, first master 1, then master 2, 3, … until master 1, and then again 2, 3, … Slaves will have network addresses higher than n. We also assume the following message stream model:

$$S_i^k = (C_i^k, T_i^k, D_i^k) \tag{2}$$

S_i^k defines a message stream i in master k (k = 1, .., n). A message stream is a temporal sequence of message

[2] In P-NET all the frame bytes are sent asynchronously, with one start bit (logical zero), 8 data bits (with LSB first), one address/data bit and one stop bit. Within a frame, a start bit must immediately follow a stop bit.

cycles concerning, for instance, the remote reading of a specific process variable. C_i^k is the longest message cycle of stream S_i^k. T_i^k is the periodicity of stream S_i^k requests. Finally, D_i^k is the relative deadline of the message cycle, that is, the maximum admissible time span between the instant when the message request is placed in the outgoing queue and the complete reception of the related response at the master's incoming queue. We consider both periodic and sporadic messages. For the case of sporadic message requests, its period corresponds to the minimum time between any two consecutive requests for that stream.

In our model the relative deadline of a message stream 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 the deadline of the first request was missed[3]. It also means that the maximum number of pending requests in the outgoing queue will be, in the worst-case, 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 the two following components: the time spent by the request in the outgoing queue, until gaining access to the bus (queuing delay); the time needed to process the message cycle, that is, to send the request and receive the related response (transmission delay)[4].

Thus,

$$R_i^k = Q_i^k + C_i^k \tag{3}$$

where Q_i^k is the worst-case queuing delay of a message stream i in a master k.

In order to have simpler and more understandable analysis, we will use the maximum token holding time (see equation (1)) for all message cycle transactions, instead of considering the actual length for each particular message cycle. Thus, we will use equation (4), instead of equation (3) to define the worst-case response time for a message request belonging to stream S_i^k:

$$R_i^k = Q_i^k + C_M \tag{4}$$

It also follows, from considering C_M instead of C_i^k (since $C_i^k \leq C_M$), that $Q_i^k = Q^k$, $\forall_i$ and $R_i^k = R^k$, $\forall_i$ [21].

3.2. Basic Analysis for the WCRT

A basic analysis for the worst-case response time can be performed if the worst-case token rotation time is assumed for all token cycles. As the token rotation time is the time span between two consecutive token visits to a particular station, the worst-case token rotation time, denoted as V, is:

$$V = n \times H \tag{5}$$

with H as defined in (1), and it gives the longest time interval between two consecutive token visits to any master k (k = 1, .., n).

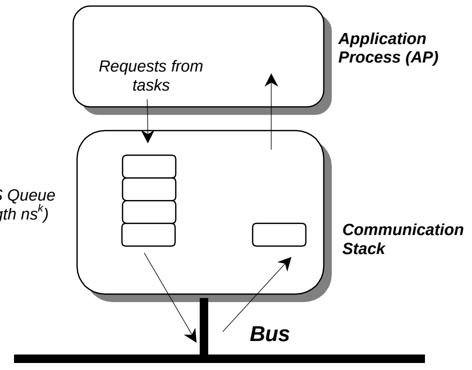

Figure 3. P-NET Architecture.

In P-NET, the outgoing queue is implemented as a FCFS queue (figure 3). Therefore, a message request can be in any position within the ns^k pending requests. ns^k is also the maximum number of requests which, at any time, are pending in the master k outgoing queue. This results from the adopted message stream model, which considers $D_i^k \leq T_i^k$. Hence, the maximum number of token visits to process a message request in a master k, is ns^k.

The worst-case queuing delay occurs if ns^k requests are placed in the outgoing queue just after a message cycle was completed (at the beginning of the token passing interval: t) and the token is fully utilised in the next ns^k consecutive token cycles.

In [21], we denoted this time instant as the Master's Critical Instant, and defined the Master's Busy Period as being the time span between the critical instant and the time instant when the last of the ns^k requests is completely processed. Based on these two definitions, we introduce the following 2 theorems.

Theorem 1 In P-NET networks, the worst-case response time of a master's message request corresponds to the longest busy period in such master.

Proof: The busy period starts when a critical instant occurs: the instant t_c. By the critical instant definition, ns^k requests are placed in the outgoing queue at the earliest possible instant. As the end of the busy period is defined as being the time instant t_e, when the last of those ns^k

[3] Actually, we can be more precise saying that deadlines will be missed if a new request appears, in the outgoing queue, before the deadline of the previous message cycle for the same message stream.

[4] As the bit rate in P-NET is 76800 bps, the propagation delay can be neglected, even for P-NET networks with length of some kilometers.

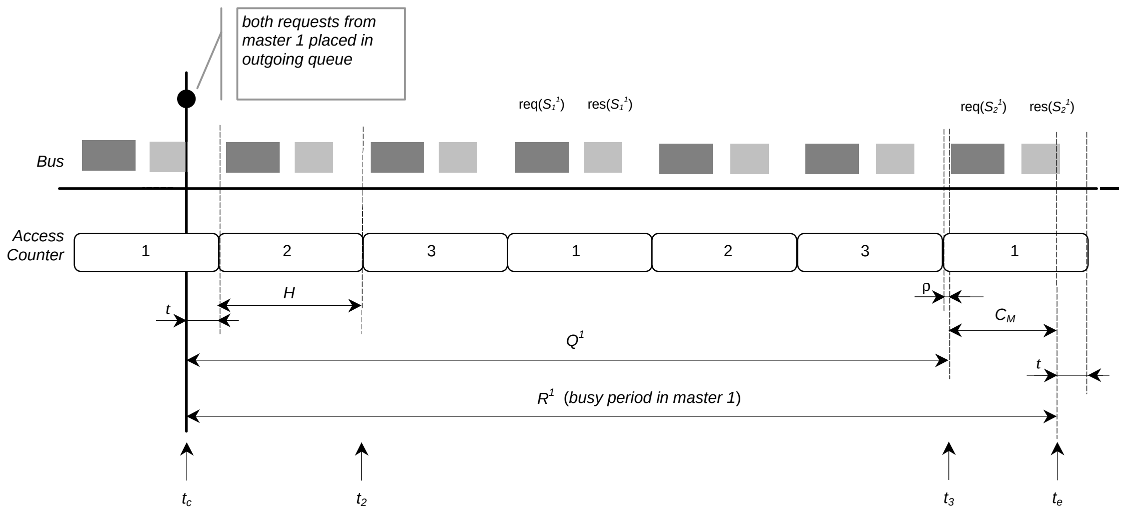

Figure 4. Example Scenario.

requests is completely processed, the difference t_c-t_e gives the worst-case response time for a message request in master k, since due to the FCFS behaviour of the outgoing queue, at t_c, a message request can be in any position, from 1st to ns^k-nd. ❑

Theorem 2 In P-NET networks, assuming that the token is fully utilised, the worst-case response time of a message request in a master k is:

$$R^k = ns^k \times V \tag{6}$$

Proof: Assuming that the token is fully utilised, the token will take t+(n-1)×H from instant t_c until the next visit to master k. At the first visit, the token arrives at t_2=t_c+ t+(n-1)×H, and only then the master will be able to process the first of the ns^k pending requests. As only one of the ns^k message requests is processed per token visit, the token will arrive at master k only at instant t_3=t_2+(ns^k-1)×V to process the last of the ns^k requests. The time elapsed since t_c is then t_3-t_c=t+(n-1)×H+(ns^k-1)×V. As the worst-case reaction time of a master is r, the last one of the ns^k message requests will start to be transmitted with a queuing delay Q^k=t+(n-1)×H+(ns^k-1)×V+r. Note that as we are assuming C_i^k=C_M, $\forall_{i,k}$, the worst-case queuing delay is equal for all message requests in the same master (Q_i^k=Q^k, $\forall_i$). As R_i^k=Q_i+C_M, the worst-case response time for a message stream i in master k is (note that R_i^k = R^k): R^k=t+(n-1)×H+(ns^k-1)×V+r+C_M, which, considering that H=r+C_M+t, can be re-written as follows: R^k=n×H+(ns^k-1)×V=V+(ns^k-1)×V=ns^k×V. ❑

Corollary In P-NET networks, assuming that the token is fully utilised, the worst-case queuing delay of a message request in a master k is:

$$Q^k = t + (n-1) \times H + (ns^k - 1) \times V + r \tag{7}$$

To illustrate both theorems 1 and 2, assume a network scenario with n = 3 and ns^1 = 2. Figure 4 shows both Q^1 and R^1 for such scenario. Note that at instant t_c, ns^1 requests are placed in the outgoing queue in an arbitrary order. Whichever the ordering, the busy period corresponds to R^1, and therefore, the worst-case response time for a message request in master 1 is (6): ns^1×V=2×V=2×3×H=6×H.

3.3. Considering the Actual Token Utilisation in the WCRT Analysis

In the previous section we derived a basic timing analysis for the evaluation of the worst-case message response time. Such analysis may however be very pessimistic, since we assumed the token as being fully utilised in the ns^k consecutive token cycles of the busy period. However, the token can only be fully utilised during that period if:

$$ns^y \geq ns^k , \forall_{y \neq k} \tag{8}$$

as, only in such case, the number of pending requests, in each master y, may be greater than ns^k. Otherwise, if $\exists_{y \neq k}$: ns^y<ns^k, the token utilisation depends on the periodicity of message streams for those masters y.

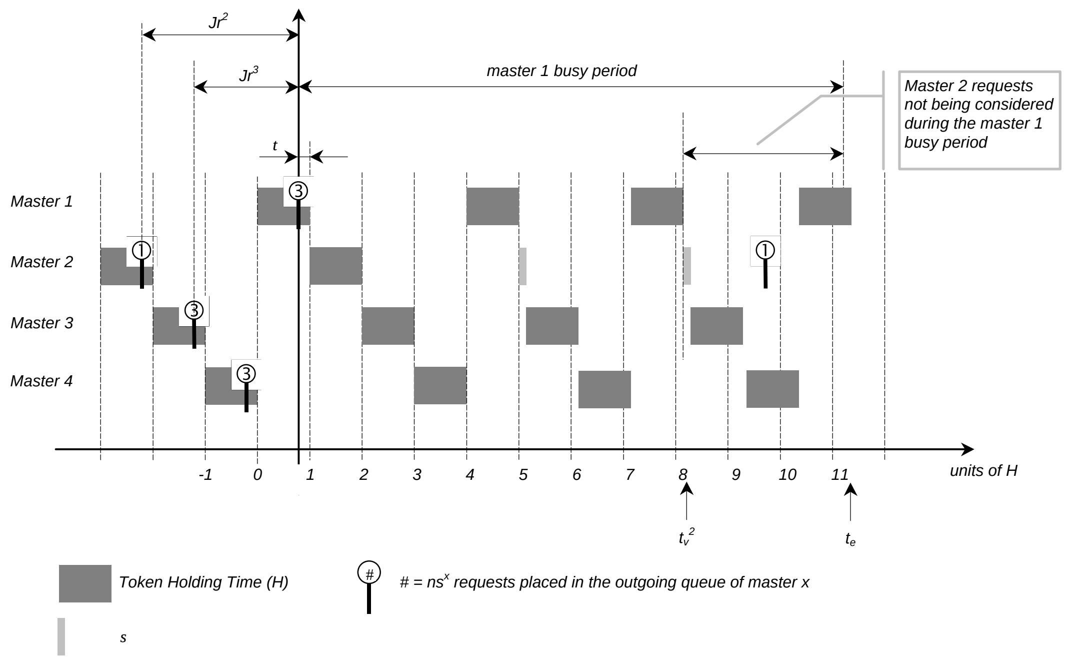

Figure 5. Example of Processing Window.

We define the Eligible Requests of Master y ($y \neq k$), as the maximum number of requests generated in that master that will be pending[5] within the busy period of master k.

Based on this definition, in [21] we proved the following P-NET theorems:

Theorem 3 In P-NET networks, the longest busy period of master k occurs when all k predecessors started their busy periods in the token cycle previous to the busy period in master k.

Theorem 4 The minimum number of unused tokens by a master y within a busy period of master k, is $Ut^y = ns^k - \min\{ns^k, ns^y + \sum_{i=1,..,ns^y} \lfloor (R^k+Ja^y)/T_i^y \rfloor\}$.

We denote $Ja^y=Jr^y-Jv^y$ as the Logical Ring Aggregate Jitter of master y in respect to master k.

Jr^y is the Logical Ring Request Jitter of master y ($Jr^y=t_r^y-t_c$), being t_r^y how much earlier than the critical instant in k, a master y can made its ns^y requests, without violating a deadline, nor processing any of those ns^y requests prior to the critical instant in master k.

Jv^y is the Logical Ring Visit Jitter of master y ($Jv^y=t_e-t_v^y$), being the time span between t_c and t_v^y ($t_v^y < t_e$) the Processing Window of Master's k Busy Period, within which, a first-positioned pending request in master y will assuredly be processed.

In a master k, the number of token cycles during the busy period is ns^k. Thus, the actual token utilisation by a master y, during the busy period of master k, is $\min\{ns^k, ns^y+\sum_{i=1,..,ns^y} \lfloor (R^k+Ja^y)/T_i^y \rfloor\}$.

Note that $ns^y+\sum_{i=1,..,ns^y} \lfloor t/T_i^y \rfloor$ gives the maximum number of requests generated by a master y within a time interval t: ns^y requests are made at the beginning of the interval, and then, new requests are made at their maximum rate. This is also known as the *asap* (as soon as possible) pattern [23].

Embodied in expression $\sum_{i=1,..,ns^y} \lfloor (R^k+Ja^y)/T_i^y \rfloor$ is that the worst-case situation appears when the ns^y requests are not simultaneously made in all masters $y \neq k$, but some time before t_c, which, for each master y, depends of its logical ring position. That quantity of time before is given by Jr^y:

$$Jr^y = [(n+k-y) \bmod n] \times H \tag{9}$$

Since $Ja^y=Jr^y-Jv^y$, also embodied in expression $\sum_{i=1,..,ns^y} \lfloor (R^k+Ja^y)/T_i^y \rfloor$ is that from the requests generated by masters $y \neq k$ during the time span of the busy period in master k, only those that appear Jv^y before the end of the busy period may be processed, with Jv^y:

$$Jv^y = [((n+k-y) \bmod n)] \times s + C_M + \sum_{\substack{i=y^{+1},..,k^{-1} \\ with \\ ns^i \geq ns^k}} (H-s) \tag{10}$$

Figure 5 illustrates these logical ring jitters when evaluating the WCRT for master 1. Note that both masters 3 and 4 have a number of streams equal to master 1. Only master 2 has fewer streams than master 1, therefore, it may not use all the 3 consecutive token cycles to process

[5] Even if they are processed during the busy interval, for a while they were pending.

message cycles. The periodicity of the only stream of master 2 is 12×H. Even a new request for that stream is made before the end of the busy period in master 1, that request will not be processed within that period, as it appeared after the last visit (within the busy interval of master 1) of the token to master 2.

It is now possible to update (6) to incorporate the actual token utilisation, considering that, for each unused token we must subtract the corresponding value of the token holding time (H), and add a s corresponding to the token passing time for the case of an unused token:

$$R^k = ns^k \times V - Ut \times (H - s) \tag{11}$$

where Ut gives the overall number of unused tokens during a busy period in master k:

$$Ut = \sum_{y=1}^{n} Ut^y \tag{12}$$

with the number of unused tokens by each master y being trivially derived from the number of eligible requests:

$$Ut^y = ns^k - \min\left\{ ns^k, ns^y + \sum_{i=1}^{ns^y} \left\lfloor \frac{\left(R^k + Ja^y\right)}{T_i^y} \right\rfloor \right\} \tag{13}$$

Therefore, equation (11) can be updated to:

$$\begin{aligned} R^k &= ns^k \times V - \\ &- \left[\sum_{y=1}^{n} \left(ns^k - \min\left\{ ns^k, ns^y + \sum_{i=1}^{ns^y} \left\lfloor \frac{R^k + Ja^y}{T_i^y} \right\rfloor \right\} \right) \right] \times (H - s) \end{aligned} \tag{14}$$

As expected, this equation embodies a mutual dependence, since R^k appears in both sides of the equation. In fact, all the previous analysis underlay this mutual dependence, since in order to evaluate R^k, Ut must be found, and *vice-versa*. The easiest way to solve equation (14) is to form a recurrence relationship [24]:

$$\begin{aligned} W^{m+1} &= ns^k \times V - \\ &- \left[\sum_{y=1}^{n} \left(ns^k - \min\left\{ ns^k, ns^y + \sum_{i=1}^{ns^y} \left\lfloor \frac{W^m + Ja^y}{T_i^y} \right\rfloor \right\} \right) \right] \times (H - s) \end{aligned} \tag{15}$$

The set {W^0, W^1, …, W^m, …} is monotonically non-decreasing, since as W evolves, less unused tokens are being considered. Starting with W^0 = 0; when W^m = W^{m+1}, the solution of equation (14) has been found.

Note that, due to the FCFS behaviour of the P-NET's pending requests queue, the WCRT will always consist of the worst-case ns^k consecutive token cycles. By considering the actual token utilisation (14) we improve the results obtained by (6). Nonetheless, a priority inversion with the length:

$$ns^k - 1 \tag{16}$$

may occur, as, in the worst-case, a message request with a more stringent deadline may stand pending in the outgoing queue in a position after the other ns^k – 1 message requests from the same master.

A priority-based dispatching policy would solve this problem: as compared to the FCFS dispatching policy, requests with more stringent deadlines would have smaller worst-case response times, as they would not wait ns^k token visits to be processed.

4. Adding Priority-Based Mechanisms to P-NET Masters

In this section we survey the main results available for single processor pre-run-time schedulability analysis for systems with tasks dispatched according to the DM priority assignment. Special emphasis will be given to the analysis for non pre-emptive systems, since the analysis for pre-emptive systems is not of much use for communicating messages in P-NET, as a message cycle can not be pre-empted.

We will then adapt these results to encompass the worst-case response time analysis of P-NET messages dispatched according to the DM priority assignment (model of figure 1).

4.1. Pre-Run-Time Schedulability Analysis of DM Dispatched Tasks

Joseph and Pandya [25] proved that the worst-case response time r_i of a task t_i is found in a scenario in which all tasks are synchronously released at instant t = 0, the critical instant, and then at their maximum rate. r_i is computed by the following recursive equation (where $hp(i)$ denotes the set of tasks of higher priority than t_i):

$$r_i^{m+1} = C_i + \sum_{j \in hp(i)} \left(\left\lceil \frac{r_i^m}{T_j} \right\rceil \times C_j \right) \tag{17}$$

The recursion ends when r_i^{m+1}=r_i^m=ri 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 D_i. If the series exceeds D_i, the task t_i is not schedulable.

This result is valid for the pre-emptive context. Fewer results are known about fixed priority-based non pre-emptive dispatching. In [24] Audsley *et al.* updated the analysis of Joseph and Pandya to include blocking factors introduced by periods of non pre-emption. The following equations represent this analysis:

$$r_i = w_i + C_i \tag{18}$$

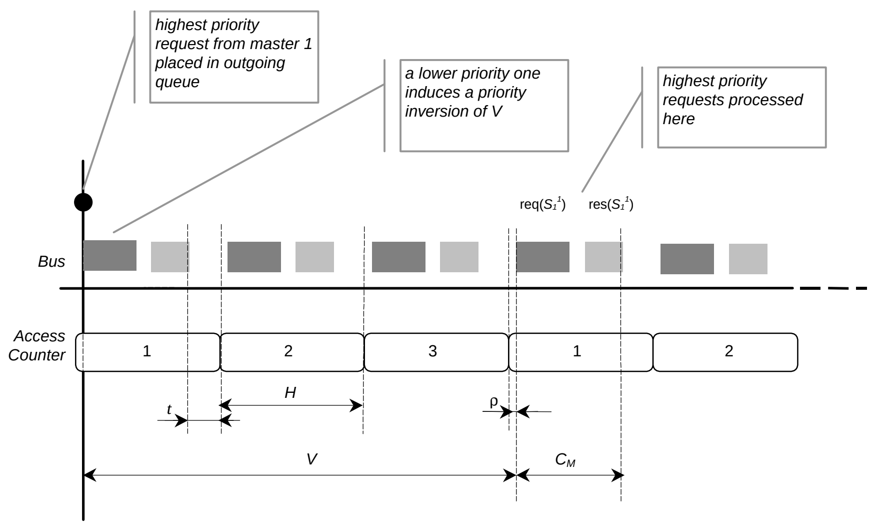
highest priority request from master 1 placed in outgoing queue
a lower priority one induces a priority inversion of V
highest priority requests processed here
req(S_1^1)
res(S_1^1)
Bus
Access Counter
1
2
3
1
2
t
H
ρ
V
C_M

Figure 6. New Assumptions for the Critical Instant.

where w_i is given by:

$$w_i^{m+1} = B_i + \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{w_i^m}{T_j} \right\rceil \times C_j \right) \tag{19}$$

B_i is the *blocking factor* of task t_i, that is, an upper bound on the time a lower priority task can execute (thus resulting in *priority inversion*) and prevent the execution of task t_i. Although the inclusion of such blocking factor was to solve the problem of non-independence between tasks, we can use this blocking factor for the schedulability analysis of non pre-emptive tasks. In this case, the blocking factor will be:

$$B_i = \max_{\forall j \in lp(i)} \{C_j\} \tag{20}$$

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

4.2. WCRT of P-NET Messages with DM Priority-Based Dispatching

In this sub-section we will adapt both equations (6) and (15) to embody the results surveyed in sub-section 4.1.

When developing the basic WCRT analysis which resulted in equation (6), we considered that all ns^k requests were placed in the outgoing queue just before the token is passed after a message cycle is processed. To adapt the results of equation (19) to the case of P-NET messages, we now consider that, for the WCRT, the *critical instant* is at the time instant when a previous pending message request is started to be processed (see figure 6). In this case, equation (6) is updated to:

$$R_i^k = B_i^k + \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{R_i^k}{T_j^k} \right\rceil \times V \right) + C_i^k \tag{21}$$

with B_i^k defined as follows:

$$\begin{cases} B_i^k = 0, \; p(i) = \min\limits_{l=1,..,ns^k} \{p(l)\} \\ B_i^k = V, \; p(i) \neq \min\limits_{l=1,..,ns^k} \{p(l)\} \end{cases} \tag{22}$$

where $p(l)$ is the priority of the message stream S_l^k (we assume that the message streams in a master have distinct priority levels).

The message stream with the highest priority will have a queuing delay of V. In order to consider the actual token utilisation, we can update equation (13) to:

$$Ut^y = \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{R_i^k}{T_j^k} \right\rceil + 1 \right) - \min\left\{ \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{R_i^k}{T_j^k} \right\rceil + 1 \right), ns^y + \sum_{i=1}^{ns^y} \left\lfloor \frac{\left(R^k + Ja^y\right)}{T_i^y} \right\rfloor \right\} \tag{23}$$

since the number of unused tokens in each master y is not the minimum value chosen from ns^k and $ns^y + \sum_{i=1,..,ns^y} \lfloor (R^k + Ja^y)/T_i^y \rfloor$ anymore, as the worst-case number of consecutive token visits to process a message requests now depends on the priority of the particular message stream of master k and is given by $\sum_{\forall j \in hp(i)} (\lceil R_i^k/T_j^k \rceil + 1)$ (in the case of the message stream with the lowest priority, we must consider only $\sum_{\forall j \in hp(i)} \lceil R_i^k/T_j^k \rceil$).

4.3. Message Release Jitter

For the analysis made in section 3, we assumed that the periodicity we were considering for message requests was the minimum interval between any two consecutive requests for that stream. This was a useful abstraction, as we were only focusing on the WCRT imposed by the P-NET's MAC.

When assuming the model (represented in figure 1) for adding priority-based dispatching mechanisms to P-NET masters, it is of more relevance to speak about application tasks, as it is the application tasks that place requests in the AP priority ordered queue.

We can assume that message requests are placed in the priority-ordered application process queue by an application task. Consequently, we can say that messages inherit from sending tasks both their period and priority level. It is implicit that tasks, at the application process level, are scheduled according to the DM priority-based policy, and most probably in a pre-emptive context. Assume also that during the execution of a message requesting task, only one request is queued.

This rises the problem of *message release jitter*, which, in the context of communication networks, has been addressed by Tindell [26] and Spuri [27]. It results from the inheritance approach that messages which are generated by requesting tasks, may have a minimum inter-arrival time smaller than the period of the tasks which generate them.

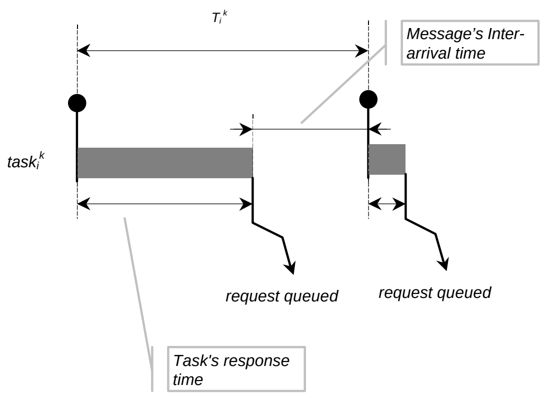

Figure 7. Message Release Jitter.

Therefore, assuming that in both equations (21) and (23) the periods of the message streams correspond to the period of the respective generating tasks, then, equation (21) must be updated to:

$$R_i^k = B_i^k + \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{R_i^k + J_j^k}{T_j^k} \right\rceil \times V \right) + C_i^k \qquad (24)$$

and equation (23) must be updated to:

$$Ut^y = \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{R_i^k + J_j^k}{T_j^k} \right\rceil + 1 \right) - \min\left\{ \sum_{\forall j \in hp(i)} \left(\left\lceil \frac{R_i^k + J_j^k}{T_j^k} \right\rceil + 1 \right), ns^y + \sum_{l=1}^{ns^y} \left\lfloor \frac{R_i^k + Ja^y + J_l^y}{T_l^y} \right\rfloor \right\} \qquad (25)$$

where J_a^b is the message release jitter of the message request of message stream *a* in master *b*. Figure 7 illustrates the concept of message release jitter. In the worst-case the message release jitter will be the worst-case response time of the generating task.

We can now define the end-to-end communication delay as:

$$E = g + R + d \qquad (26)$$

g represents the worst-case generation delay for a master application task to generate and queue a specific message request. This would also correspond to the message release jitter, which will be used for computing R_i^k (the WCRT of a P-NET message request). Finally, *d* represents the delivery delay, that is, the worst-case response time of the receiving task (is in the same host processor as the sending task) to process the response.

For the computation of the worst-case response time of the sending tasks (and receiving tasks), the consideration of offsets [28,29] can be used to model each pair of sending and receiving task as a transaction, hence reducing the end-to-end response time analysis. Furthermore, using the results from [29], the sending and receiving tasks can actually be modelled as being the same task, which suspends itself by the worst-case message response time.

5. Conclusions

In this paper we have drawn a comprehensive study on how to use P-NET to support real-time communications in distributed computer controlled systems. We surveyed the previous analysis for the worst-case response time (WCRT) in P-NET networks. Even by considering the actual token utilisation during a busy period, the WRCT is still very much penalised by the first-come-first-served (FCFS) policy that P-NET uses at the data link layer. Hence, we proposed a model for implementing fixed priority-based dispatching mechanisms at each master's application process level. In this way, we are able to support distributed computer controlled systems with tighter deadlines.

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] Sha, L., Rajkumar, R., Lehoczky, J.: “Real-Time Applications Using IEEE Futurebus+”, in IEEE Micro, June 1990.

[3] Jeong, S., Kim, Y., Kwon, W.: “Point-to-Point Real-Time Communication Over Backplane Bus”, in Proceedings of the 15th IFAC Workshop on Distributed Computer Control Systems, September 1998.

[4] ISO TC184/SC5/WG1, A Reference Model for Discrete Parts Manufacturing, Technical Report No 58, International Organisation for Standardisation 1988.

[5] Decotignie, J.-D., Pleinevaux, P., “A Survey on Industrial Communication Networks”, in Annales des Télécommunications, invited paper, Vol. 48, No. 9-10, pp. 435-448, 1993.

[6] Normes FIP NF C46-601 to NF C46-607, Union Technique de l'Electricité, AFNOR, 1990.

[7] Profibus Standard DIN 19245 part I and II. Translated from German, Profibus Nutzerorganisation e.V., 1992.

[8] SAE J1583, Controller Area Network (CAN), an In-Vehicle Serial Communication Protocol. SAE Handbook, Vol. II, 1992

[9] The P-NET Standard. International P-NET User Organisation ApS, 1994.

[10] General Purpose Field Communication System, Vol. 1/3 (P-NET), Vol. 2/3 (Profibus), Vol. 3/3 (FIP), CENELEC, 1996.

[11] Pedro, P., Burns, A.: “Worst Case Response Time Analysis of Hard Real-Time Sporadic Traffic in FIP Networks”, in Proceedings of 9th Euromicro Workshop on Real-time Systems, Toledo, Spain, pp. 5-12, 1997

[12] Raja, P., Ruiz, L., Decotignie, J.-D.: “On the Necessary Real-Time Conditions for the Producer-Distributer-Consumer Model”, In Proceedings of 1st IEEE Workshop on Factory Communication Systems (WFCS'95), Leysin, Switzerland, 1995.

[13] Grow, R.: “A Timed Token Protocol for Local Area Networks”, Proceedings of Electro’82, Token Access Protocols, Paper 17/3, May 1982.

[14] 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.

[15] Montuschi, P., Ciminiera, L., Valenzano, A.: “Time Characteristics of IEE802.4 Token Bus Protocol”, IEE Proceedings, 139 (1), pp. 81-87, January 1992.

[16] Tovar, E., Vasques, F., “Guaranteeing Real-Time Message Deadlines in Profibus Networks”, in proceedings of the 10th Euromicro Workshop on Real-time Systems, Berlin, Germany, June 1998.

[17] Tovar, E., Vasques, F., “Setting Target Rotation Time in Profibus Based Real-Time Distributed Applications”, in proceedings of the 15th IFAC Workshop on Distributed Computer Control Systems (DDCS'98), Como, Italy, September 1998.

[18] Tindell, K., Hansson, H., Wellings, A., “Analysing Real-Time Communications: Controller Area Network (CAN)”, in Proceedings of the IEEE Real-Time Systems Symposium, pp. 259-263, December 1994.

[19] Tindell, K., Burns, A., Wellings, A., “Analysis of Hard Real-Time Communications”, in Journal of Real-Time Systems, No. 9, 1995.

[20] Tovar, E., Vasques, F., Burns, A., “Supporting Real-Time Distributed Computer-Controlled Systems with Multihop P-NET Networks”, submitted to Control Engineering Practice, September 1998.

[21] Tovar, E., Vasques, F., Burns, A., “Communication Response Time in P-NET Networks: Worst-Case Analysis Considering the Actual Token Utilisation”, Submitted to the Journal of Real-Time Systems, December 1998. Available as Technical Report YCS-99-312, from the University of York. (http://www.cs.york.ac.uk/rts).

[22] Jenkins, C., “P-NET as a European Fieldbus Standard EN 50170 vol. 1”, in the Institute of Measurement + Control Journal, 1997.

[23] Liu, C., Layland, J., “Scheduling Algorithms for Multiprogramming in Hard-Real-Time Environment”, in Journal of the ACM, 20(1), pp. 46-61, 1973.

[24] Audsley, N., Burns, A., Richardson, M., Tindell, K., Wellings, J., “Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling”, in Software Engineering Journal, 8(5), pp. 284-292, 1993.

[25] Joseph, M., Pandya, P., “Finding Response Times in a Real-Time System”, The Computer Journal, Vol. 29, NO. 5, pp. 390-395, 1986.

[26] Tindell, K., “Holistic Schedulability Analysis for Distributed Hard Real-Time Communications”, University of York, Technical Report YCS-93-197, 1993.

[27] Spuri, M., “Holistic Analysis for Deadline Scheduled Real-Time Distributed Systems”, INRIA, Technical Report No. 2873, 1996.

[28] Tindell, K., “Adding Time-Offsets to Schedulability Analysis”, Technical Report YCS 221, Dept. of Computer Science, University of York, 1994.

[29] Palencia, J., Harbour, M., “Schedulability Analysis of Tasks with Static and Dynamic Offsets”, in Proceedings of IEEE Real-Time Systems Symposium, pp. 26-37, December 1998.