Technical Report

Analysing TDMA with Slot Skipping

Björn Andersson
Eduardo Tovar
Nuno Pereira

TR-051201
Version: 1.0
Date: December 2005

Analysing TDMA with Slot Skipping

Björn ANDERSSON, Eduardo TOVAR, Nuno PEREIRA

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: {bandersson, emt, nap}@dei.isep.ipp.pt

http://www.hurray.isep.ipp.pt

Abstract

We propose a schedulability analysis for a particular class of time division multiple access (TDMA) networks, which we label as TDMA/SS. SS stands for slot skipping, reflecting the fact that a slot is skipped whenever it is not used. Hence, the next slot can start earlier in benefit of hard real-time traffic. In the proposed schedulability analysis, we assume knowledge of all message streams in the system, and that each node schedules messages in its output queue according to a rate monotonic policy (as an example). We present the analysis in two steps. Firstly, we address the case where a node is only permitted to transmit a maximum of one message per TDMA cycle. Secondly, we generalise the analysis to the case where a node is assigned a budget of messages per TDMA cycle it may transmit. A simple algorithm to assign budgets to nodes is also presented..

Analysing TDMA with Slot Skipping

B. Andersson, E. Tovar and N. Pereira
IPP Hurray Research Group
Polytechnic Institute of Porto, Portugal
{bandersson,emt,npereira}@dei.isep.ipp.pt

Abstract

We propose a schedulability analysis for a particular class of time division multiple access (TDMA) networks, which we label as TDMA/SS. SS stands for slot skipping, reflecting the fact that a slot is skipped whenever it is not used. Hence, the next slot can start earlier in benefit of hard real-time traffic. In the proposed schedulability analysis, we assume knowledge of all message streams in the system, and that each node schedules messages in its output queue according to a rate monotonic policy (as an example). We present the analysis in two steps. Firstly, we address the case where a node is only permitted to transmit a maximum of one message per TDMA cycle. Secondly, we generalise the analysis to the case where a node is assigned a budget of messages per TDMA cycle it may transmit. A simple algorithm to assign budgets to nodes is also presented.

1. Introduction

A fundamental problem in distributed real-time systems is the sharing of a communication resource between message streams on different nodes such that real-time requirements are satisfied. TDMA (time division multiple access) communication protocols solve this by assigning messages to time slots in a way that no two nodes transmit at the same time and messages' queuing delays are bounded. Typically, these communication protocols operate on the basis of TDMA cycles, where a node is assigned one or many time slots. Usually, each slot has a fixed length and the number of slots per cycle is also fixed. Hence, a TDMA cycle has fixed and known time duration, and upper bounds on messages' queuing delays can be proved.

The majority of research work on TDMA communications addresses the problem of finding appropriate schedules (TDMA frames/templates) for guaranteeing timeliness to real-time message streams. This is the case of analysis over time-triggered protocols such as TTP [1]. It is also the case of works addressing distance constraints (maximum timing interval between two adjacent instances of the same message stream) as an additional temporal restriction [2, 3]. Unfortunately, the flexibility in assigning time slots to nodes in these approaches comes at a price: an unused slot is wasted and cannot be used for any other hard real-time traffic. A message stream with periodic messages may need a specific time slot in a TDMA cycle only in a few cycles, while in the majority of the cycles that time slot is not used, hence wasted. One way to overcome this waste is to have a larger TDMA cycle serving several instances of a message stream. Unfortunately, in the extreme case, the length of a TDMA cycle may need to be the least common multiple of periods, to avoid wasted slots.

In contrast, however, consider TDMA protocols with slot skipping (TDMA/SS); that is, a slot is skipped when it is not used. Hence, the next slot can start earlier in benefit of hard real-time traffic. Such an approach is already in use in some commercial-off-the-shelf (COTS) technology [4], but in order to be useful, a schedulability analysis that takes slot skipping into account is needed. Such an analysis is still missing for a generic TDMA/SS network.

In this paper we present a schedulability analysis for TDMA networks with slot skipping (TDMA/SS). We assume that all message streams in the system are known, and that each node schedules messages in its output queue according to a rate monotonic policy (as an example). We present the analysis in two steps. Firstly, we address the case where a node is only permitted to transmit one message per TDMA cycle. Secondly, we generalise the analysis to the case where a node is assigned a budget of messages it may transmit per TDMA cycle. Clearly, the queuing times of messages depend on the assignment of budgets to nodes. In fact, a poor assignment can cause deadlines to be missed, even if the utilisation is arbitrarily low. We propose a simple and suitable algorithm to assign budgets to nodes.

We consider this research work to be significant for two reasons. First, our analysis is tighter than any other previous analysis on TDMA networks that skip slots [5]. Second, we also consider the case where a node can be assigned a fixed number of slots, whereas previous work only considered the case of a single slot per TDMA cycle.

The remainder of this paper is organised as follows. Section 2 illustrates the basics of operation of the TDMA/SS network. In Section 3, we reason and present a methodology on how to compute accurate message queuing delays in a network where a node is only permitted to transmit a single message per TDMA cycle (the SMTC case). This analysis is then extended, in Section 4, to networks that allow multiple messages per TDMA cycle (the MMTC case). Section 5 presents a numerical example to illustrate the use of the analysis. Section 6 compares our approach to other approaches in real-time communications. Finally, in Section 7, conclusions are drawn.

2. TDMA Networks with Slot Skipping

2.1. Network and Message Models

Our network is composed of n nodes, communicating messages via a shared medium. Contention access between nodes is resolved by a time division multiple access (TDMA) control schema. The access to the medium is ordered by time, such that each node is assigned one or more time slots, each of length T_{MS}, in a cyclic schedule – the TDMA cycle. When a node observes its turn to access the shared medium, it may transmit messages up to the number of time slots assigned to it. To signal that the node will not transmit any more messages during the current TDMA cycle, a node transmits a protocol slot of length T_{PR} (typically $T_{PR} << T_{MS}$). In a concrete setting, nodes can implement this protocol slot simply by staying silent during a T_{PR} time span.

Our network model can be described as follows:

$$net = \left(n, \left\{N^1, N^2, \ldots, N^n\right\}, T_{MS}, T_{PR}\right) \quad (1)$$

Associated to each node k (k ranging fom 1 to n), there is a set $\{S_1^k, S_2^k, \ldots, S_{ns^k}^k\}$ of ns^k message streams. A node k is permitted to transmit at most mpc^k (messages per cycle) in a TDMA cycle. Hence, a node k is defined as follows:

$$N^k = \left(ns^k, \left\{S_1^k, S_2^k, \ldots, S_{ns^k}^k\right\}, mpc^k\right) \quad (2)$$

A message stream with index i (i ranging from 1 to ns^k) associated to node k is denoted as S_i^k. Each message stream is characterised by T_i^k and D_i^k. T_i^k is the periodicity at which a message related to S_i^k is queued to be transmitted to the network. D_i^k is the relative deadline of S_i^k.

Every message needs to be queued before being transmitted. We consider the use of rate monotonic (RM) scheduling [6] in all network nodes to serve the output queue of message streams. Let q_i^k denote the maximum queuing time of messages belonging to S_i^k. Let r_i^k denote the maximum response time of all messages belonging to S_i^k, $r_i^k = q_i^k + T_{MS}$. If $r_i^k \le D_i^k$ then we say that S_i^k meets its deadlines. We are interested in finding out whether all messages meet their deadlines. In order to do so, we will find an upper bound on q_i^k. This upper bound is denoted Q_i^k. Let R_i^k denote an upper bound on the response time; that is, $R_i^k = Q_i^k + T_{MS}$. If $R_i^k \le D_i^k$ then we say that S_i^k is deemed to meet its deadlines according to our analysis technique.

Our analysis assumes that $D_i^k \le T_i^k$. Therefore, a message from S_i^k must finish its transmission before a new message from S_i^k arrives to the node's output queue. We assume that all messages in the network have the length T_{MS}.

In the description of the TDMA/SS protocol and related time analysis, some shorthand notations are useful. The next and the previous nodes are denoted as follows:

$$prev(k) = \begin{cases} n, & \text{if } k = 1 \\ k-1, & \text{if } 2 \le k \le n \end{cases}$$
$$next(k) = \begin{cases} k+1, & \text{if } 1 \le k \le n-1 \\ 1, & \text{if } k = n \end{cases} \quad (3)$$

Additionally, and since we assume RM to be used to schedule messages in the node's output queue, the set of higher/lower-priority message streams are denoted as follows:

$$hp^k\left(S_i^k\right) = \left\{ S_j^k : \left(T_j^k < T_i^k\right) \vee \left(T_j^k = T_i^k \wedge j < i\right) \right\}$$
$$lp^k\left(S_i^k\right) = \left\{ S_j^k : \left(S_j^k \notin hp^k\left(S_i^k\right)\right) \wedge \left(S_j^k \neq S_i^k\right) \right\} \quad (4)$$
$$TMIN = \min_{k=1..n}\left(\min_{\forall S_i^k \, on \, N^k} T_i^k \right)$$

where TMIN is also introduced to denote the minimum period among all the message streams in the system.

We will now describe the operation of the network protocol being used. During the operation of the protocol, all nodes maintain a variable – `address_counter` – that keeps track of the node holding the right to transmit at any time. `address_counter` has the same value on all nodes, and thus in the discussion we treat it as a variable. When `address_counter` makes the transition to k, then node k will dequeue and transmit up to mpc^k messages from its output queue. If the output queue contains $0 \le x < mpc^k$ messages, then only those $0 \le x$ messages are transmitted (we say that node k skips mpc^k - x slots). After the transmission of those x

messages, a protocol slot is transmitted (this takes T_{PR} time units). As a consequence, the above mentioned system-wide variable will change as follows: `address_counter := next(address_counter)`.

When a node does not transmit, it listens to the network to update `address_counter` consistently with the other nodes. For this, we assume that all nodes hear the same state of the network.

2.2. Network Example and Operation

As an instantiation of (1), (2) and (3) concerning the previously described network and message models, consider a network with 3 nodes as follows:

$$net = \left(3, \left\{N^1, N^2, N^3\right\}, 1, 1/5\right)$$

$$\begin{cases} N^1 = \left(3, \left\{S_1^1, S_2^1, S_3^1\right\}, 1\right) \\ T_1^1 = D_1^1 = 4.0 \\ T_2^1 = D_2^1 = 13.0 \\ T_3^1 = D_3^1 = 13.4 \end{cases} \quad \begin{cases} N^2 = \left(1, \left\{S_1^2\right\}, 1\right) \\ T_1^2 = D_1^2 = 5.2 \end{cases} \quad \begin{cases} N^3 = \left(1, \left\{S_1^3\right\}, 1\right) \\ T_1^3 = D_1^3 = 7.0 \end{cases}$$

Figure 1. Example network scenario.

Consider that the arrival pattern of messages to the output queues is as illustrated in Figure 2a. For this scenario, the timeline for message transmissions and address counter evolution in the network is as illustrated in Figure 2b.

The events at time 0 require further explanation. We are assuming that:

1. a message from S_3^1 arrives marginally before time 0;
2. the `address_counter` changes from 3 to 1 at time 0;
3. and messages from both S_1^1 and S_2^1 arrive at time 0.

We also assume that a message is only able to be transmitted by node k, if and only if it has been queued before `address_counter` changes to the value k. As a result, and for the exemplified scenario, neither the message from S_1^1 nor the one from S_2^1 are transmitted at time 0. Instead, a message from S_3^1, which has lower priority, is transmitted at time 0, since this was the only message ready in the output queue of node 1 at the time `address_counter` changes to 1.

Looking now at the scheduling at time $t > 0$, observe that every time a message is transmitted it takes 1 time unit, and after there is a protocol slot of 1/5 time units. However, in some of the illustrated TDMA cycles, only a protocol slot is transmitted. This occurs because, at the time the node was granted the right to transmit, its output queue was empty (for example, the output queue of node 2 is empty at time instant 4.8).

Consider the message of S_2^1 that was placed in the output queue at time 0. This message is queued during [0,10.4) and hence q_2^1 is 10.4. The message of S_2^1 is blocked during the time interval [0,3.6) because some messages, a lower priority message S_3^1 and other messages S_1^2 and S_1^3, cause S_2^1 to be queued although it has the higher priority. The message of S_2^1 suffers from interference during [3.6,10.4).

In order to see why the schedulability analysis of this system is non-trivial, look at time instant 10. At this time instant, a message from S_2^1 (queued at time 0) is still in the output queue, and a message from another message stream, S_1^2, arrives. However, this message from S_1^2 does not have any effect on the queuing time of the message from S_2^1, transmitted at time instant 10.4.

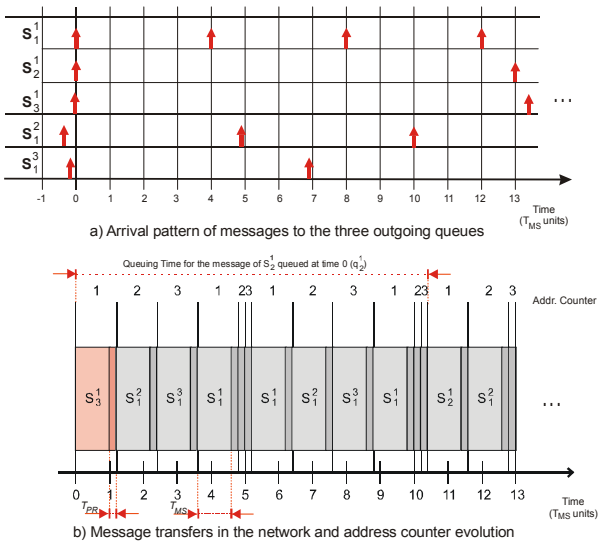

Figure 2. Arrival times and schedule of the example network scenario.

3. Single Message per TDMA Cycle (SMTC)

In this section, we will develop an accurate schedulability analysis technique for the network described in Section 2, considering the case of a single message per TDMA cycle (SMTC); that is, $\forall\, k : mpc^k = 1$. Response time equations [7] for static-priority scheduling on a uniprocessor can be extended to the problem of finding the queuing delay in communication networks. Inspired by this, we can reason as follows. If nodes never skip slots, then we can compute the queuing delay Q_i^k of a message as follows:

$$Q_i^k = B_i^k + \sum_{S_j^k \in hp^k(S_i^k)} \left(\left\lceil \frac{Q_i^k}{T_j^k} \right\rceil \times T_{TDMA} \right) \qquad (5)$$

where T_{TDMA} corresponds to the maximum TDMA cycle duration; that is, the maximum time interval that can elapse between two consecutive node accesses to the network. In our network model, this quantity is given by:

$$T_{TDMA} = n \times T_{MS} + n \times T_{PR} \qquad (6)$$

The blocking B_i^k can be computed as follows:

$$B_i^k = \begin{cases} n \times T_{MS} + n \times T_{PR}, & \text{if } lp^k(S_i^k) \neq \varnothing \\ (n-1) \times T_{MS} + n \times T_{PR}, & \text{if } lp^k(S_i^k) = \varnothing \end{cases} \quad (7)$$

Equation (7) requires some further explanation. If lp^k $(S_i^k) \neq \varnothing$ then the message from S_i^k is not the lowest-priority message at the output queue of node k. If `address_counter` has just made the transition to k and a message from S_i^k arrives marginally later, then S_i^k will have to wait until the `address_counter` becomes k again. This takes $n \times T_{MS} + n \times T_{PR}$ time units. If lp^k $(S_i^k) = \varnothing$ then S_i^k is the lowest-priority message at the output queue of node k. If `address_counter` has just made the transition to k, the output queue of node k was empty and messages from all message streams arrive marginally later at node k, then messages from S_i^k will have to wait until address counter becomes k again. This takes $(n - 1) \times T_{MS} + n \times T_{PR}$ time units. The reason is that it has to wait T_{PR} time units for the address counter to become $next(k)$, and then it has to wait an additional amount of $(n - 1) \times T_{MS} + (n - 1) \times T_{PR}$ time units.

We will now compute Q_i^k considering the effect of slot skipping. Equation (5) can be refined as follows:

$$Q_i^k = B_i^k + \left(\sum_{S_j^k \in hp^k(S_i^k)} \left\lceil \frac{Q_i^k}{T_j^k} \right\rceil \right) \times T_{TDMA} - \left[\sum_{y=1, y \neq k}^{n} nss^{y \to k}(Q_i^k, i) \right] \times T_{MS} \quad (8)$$

where $nss^{y \to k}(Q_i^k, i)$ denotes a lower bound on the number of skipped slots on node y during a time interval of length Q_i^k. The term T_{MS} represents the amount of time saved when a slot is skipped.

We will compute $nss^{y \to k}(Q_i^k, i)$ by considering how many TDMA cycles a message belonging to S_i^k has to wait in the output queue before being transmitted. The number of skipped slots on node y is the difference between the number of slots that were available to node y and the actual number of slots used by node y. Computing these quantities is however not trivial, and therefore we will use upper and lower bounds on them. A quantity that starts with LB stands for a lower bound and, analogously, UB stands for an upper bound. Using these bounds and observing that any lower bound on the number messages must be non-negative, we obtain:

$$\begin{aligned} &LBnumber\ of\ unused\ slots\ on\ N^y = \\ &\max\{0, LBnumber\ of\ slots\ that\ were\ available\ to\ node\ y - \\ &\quad UBnumber\ of\ slots\ that\ was\ transmitted\ by\ node\ y\} \end{aligned} \quad (9)$$

Since only one message is transmitted per TDMA cycle, we know that this is also the number of messages transmitted on node y. Based on this reasoning, one may believe that a lower bound on the number of skipped slots on node y during a time interval of length W_i^k is:

$$\max\left\{0, \sum_{S_j^k \in hp^k(S_i^k)} \left\lceil \frac{W_i^k}{T_j^k} \right\rceil - \left(ns^y + \sum_{\forall S_j^y on\ N^y} \left\lfloor \frac{W_i^k}{T_j^y} \right\rfloor \right)\right\} \quad (10)$$

However, equation (10) is used only to provide some insight on the reasoning towards the final results. It is incomplete because we need to assign a value to W_i^k. It would be tempting to use $W_i^k = Q_i^k$. But, unfortunately, doing so would not be correct, because the maximum queuing delay (or minimum number of skipped slots) does *not* occur when all messages on all nodes arrive at the same time. Additionally, some messages that arrive late on node y do not affect node k.

We will now compute a correct upper bound on the queuing delay when slot skipping is considered. Let t_0 denote the time instant when a message of S_i^k of maximum queuing time arrives. Consider the message stream S_j^y on node y, with $y = prev(k)$. This message stream S_j^y has a message which arrived before t_0 or at t_0. Let us call it M. At which time should M arrive to generate the maximum number of transmissions that cause a delay on the message from S_i^k? It should arrive late enough to make sure that its entire transmission time T_{MS} occurred after t_0 or at t_0, but it should arrive as early as possible to maximise the number of transmissions of S_j^y that cause a delay on the message from S_j^k. This occurs when M arrives at time $t_0 - T_{PR}$.

We can repeat this argument with node $prev(prev(k))$, $prev(prev(prev(k)))$, and so on. Hence, we obtain the following expression to compute the number of skipped slots:

$$\max\left\{0, \sum_{S_j^k \in hp^k(S_i^k)} \left\lceil \frac{Q_i^k}{T_j^k} \right\rceil - \left(ns^y + \sum_{\forall S_j^y on\ N^y} \left\lfloor \frac{Q_i^k + \Phi^{y \to k}}{T_j^y} \right\rfloor \right)\right\} \quad (11)$$

where $\Phi^{y \to k}$ is given as follows:

$$\Phi^{y \to k} = \begin{cases} 0 & \text{if } y = k \\ T_{PR} + \Phi^{next(y) \to k} & \text{if } y \neq k \end{cases} \quad (12)$$

Although (11) can be used to compute a lower bound on the number of messages on node y, we will not do so since there is a source of pessimism that we will reduce first. In order to understand this, consider Figure 2 illustrating the operation of the protocol. Let us try to compute Q_2^1. In that TDMA cycle, a message from node 3 will be processed before Q_2^1 if it arrives at node 3's output queue at least T_{PR} before the end of the time window Q_2^1 (if the message would arrive later, then `address_counter` would have already changed to 1). This reasoning can be extended so that a message on node 2 that should be transmitted before the end of the time window Q_2^1 must arrive $2 \times T_{PR}$ before

the end of the time window. This reasoning applies regardless of whether node 2 or node 3 transmits at the end of the window Q_2^1. If, however, node 3 transmits a message, and another message (belonging to S_1^2) arrives on node 2, that message from S_1^2 must arrive T_{MS} + $2 \times T_{PR}$ time units before the end of the window Q_2^1.

We will now present the general equations to compute the window of a node y. These windows are used to compute the number of skipped slots that are generated at node y.

It turns out that finding how much the window should be shrunken is difficult, and therefore, analogously to a previous reasoning, we will instead find a lower bound on how much the window should be shrunken. Clearly this offers an upper bound on the length of the window, and so this is safe.

Let $\Omega^{y \to k}$ denote a lower bound on the amount that the window of node y should be shrunken at the end of the Q_i^k. In that way we have:

$$\Omega^{y\to k}(t)=\begin{cases}0, & if \left\langle\ y=k\right.\\ T_{MS}+T_{PR}+\Omega^{next(y)\to k}(t), & if \left\langle\begin{array}{l}y\neq k\ \ and\\ LBql^{y\to k}(t)\geq 1\end{array}\right.\\ T_{PR}+\Omega^{next(y)\to k}(t), & if \left\langle\begin{array}{l}y\neq k\ \ and\\ LBql^{y\to k}(t)<1\end{array}\right.\end{cases} \quad (13)$$

Intuitively, we can understand (13) as follows. If $y = k$ then we are looking at the skipped slots on the node where the message from S_i^k is assigned. This means that the window should not be shrunken at all and thus $\Omega^{y \to k}$ should be zero. Otherwise, y and k are different nodes. Then, it matters if node y transmitted a message at the end of the window Q_i^k. If it did, then the window of node y should finish T_{MS} + T_{PR} earlier than node *next*(y). In order to know if a message was transmitted on node y at the end of the window Q_i^k, we might compute the length of the queue of output messages at node y. Finding if a message is transmitted is hard however, and thus we will use a lower bound instead. In such way, if the lower bound on the queue length is 1 or greater, then we know that a message was transmitted. $LBql^{y \to k}$ denotes this lower bound and it stands for (lower-bound queue-length). We compute it as follows:

$$LBql^{y\to k}(t)=\left(\sum_{\forall S_j^y\ on\ N^y}\left\lfloor\frac{L^{y\to k}(t)}{T_j^y}\right\rfloor\right)-\left(\sum_{\forall S_j^k\ on\ N^k}\left\lceil\frac{L^{y\to k}(t)}{T_j^k}\right\rceil\right) \quad (14)$$

where

$$L^{y\to k}(t)=\max\left\{0,\ t-\Omega^{next(y)\to k}(t)-(T_{MS}+T_{PR})\right\} \quad (15)$$

It may appear that there is a circular dependency between (13) and (15) since to compute Ω we need to know the value of Ω. There is however no such dependency. We can compute $\Omega^{k \to k}$ easily from (13). We can compute $\Omega^{prev(k) \to k}$ from (13) as well; it depends on $\Omega^{k \to k}$, which we have already computed. We can compute $\Omega^{prev(prev(k)) \to k}$ from (13) in the same way; it depends on $\Omega^{prev(k) \to k}$, which we have already computed too. Hence, we can compute any Ω with no circular dependency. We will omit the proof of (14), because it is a special case of an inequality that we will use in Section 4, about multiple messages per TDMA cycle (MMTC). Based on (11), we obtain the following result:

$$nss^{y\to k}(t,i)=\max\left\{0,\ \sum_{S_j^k\in hp^k(S_i^k)}\left\lceil\frac{t}{T_j^k}\right\rceil-\left(ns^y+\sum_{\forall S_j^y\ on\ N^y}\left\lfloor\frac{t+\Phi^{y\to k}-\Omega^{y\to k}(t)}{T_j^y}\right\rfloor\right)\right\} \quad (16)$$

where $\Phi^{y \to k}$ is as defined in (12) and $\Omega^{y \to k}$ is as defined in (13).

4. Multiple Messages per TDMA Cycle (MMTC)

If nodes are very unequally loaded, then some nodes will have many skipped slots while others will be busy most of the time. When nodes are idle, they still consume T_{PR} time units of the network. This is an overhead. It would be desirable that a node is only given TDMA slots if it has something to transmit. We will now turn our attention to the case of multiple messages per TDMA cycle (MMTC), where mpc^k (messages per cycle) is permitted to be greater than 1. It reduces the overhead and hence it offers a greater ability to meet deadlines. To understand this, consider the following simple scenario (Figure 3).

$$net=\left(2,\left\{N^1,N^2\right\},1,1/5\right)$$

$$\begin{cases}N^1=\left(72,\left\{S_1^1,S_2^1,\ldots,S_{72}^1\right\},1\right)\\ \quad T_1^1=D_1^1=100.0\\ \quad T_2^1=D_2^1=100.0\\ \quad \cdots\\ \quad T_{72}^1=D_{72}^1=100.0\end{cases}\qquad \begin{cases}N^2=\left(1,\left\{S_1^2\right\},1\right)\\ T_1^2=D_1^2=100.0\end{cases}$$

Figure 3. Message streams that need MMTC.

For this scenario, with SMTC we have $mpc^1 = 1$ and $mpc^2 = 1$. Let us analyse S_{72}^1, the message stream in node 1 with the lowest priority. It will have to wait for at least $71 \times (T_{MS} + T_{PR}) + 70 \times T_{PR}$ until it is permitted to transmit. It will finish its transmission no earlier than time $71 \times (T_{MS} + T_{PR}) + 70 \times T_{PR} + T_{MS} = 100.2$, thus missing its deadline. But, if we use $mpc^1 = 72$ and $mpc^2 = 1$, then deadlines would be met.

This overhead becomes more and more severe the larger the network is. Actually, one can extend the previous example to show that there is a set of message

streams (all with the same period) such that the utilisation of the network approaches zero and a deadline is missed if SMTC is used, while all deadlines are met with MMTC. Assigning $mpc^k > 1$ may not only reduce the overhead; it may also change the schedule favourably, and hence $mpc^k > 1$ may be useful even if $T_{PR} = 0$. Motivated by this, we will first present an extension of our single message per TDMA cycle analysis, and then propose a heuristic on how to choose the *mpc* value for each network node.

4.1. Analysis

One obvious difference with the MMTC is that the TDMA cycle duration must now be computed as follows:

$$T_{TDMA} = \left(\sum_{l=1}^{n} mpc^l \right) \times T_{MS} + n \times T_{PR} \quad (17)$$

while the blocking B_i^k is given by:

$$B_i^k = \left[\left(\sum_{l=1, l \neq k}^{n} mpc^l \right) + \min \left\{ mpc^k, \left| lp^k \left(S_i^k \right) \right| \right\} \right] \times T_{MS} + n \times T_{PR} \quad (18)$$

We can adapt the SMTC equation to compute the queuing time in the case of MMTC. If node k needs to transmit x messages, it takes $\lfloor x / mpc^k \rfloor$ TDMA cycles, and it also needs to wait for $x \bmod mpc^k$ message slots. Therefore, an upper bound on the queuing delay can be computed as follows:

$$Q_i^k = B_i^k + T_{TDMA} \times \left\lfloor \frac{\sum_{S_j^k \in hp^k(S_i^k)} \left\lceil \frac{Q_i^k}{T_j^k} \right\rceil}{mpc^k} \right\rfloor + \left(\left(\sum_{S_j^k \in hp^k(S_i^k)} \left\lceil \frac{Q_i^k}{T_j^k} \right\rceil \right) \bmod mpc^k \right) \times T_{MS} - \left[\sum_{y=1, y \neq k}^{n} nss^{y \to k} \left(Q_i^k, i \right) \right] \times T_{MS} \quad (19)$$

Computing the number of skipped slots can be made similarly to the SMTC case, but some of the terms require more care.

$$nss^{y \to k}(t, i) = \max \left[0, \left\lfloor \frac{\sum_{S_j^k \in hp^k(S_i^k)} \left\lceil \frac{t}{T_j^k} \right\rceil}{mpc^k} \right\rfloor \times mpc^y - \left(ns^y + \sum_{\forall S_j^y \text{ on } N^y} \left\lfloor \frac{t + \Phi^{y \to k} - \Omega^{y \to k}(t)}{T_j^y} \right\rfloor \right) \right] \quad (20)$$

The terms $\Phi^{y \to k}$ and $\Omega^{y \to k}$ have the same interpretation as in the SMTC case, but we will revisit their equations now. The term $\Phi^{y \to k}$ does not change, and the intuition behind it is the same as the one for the SMTC case.

We will now present and prove the equations for computing $\Omega^{y \to k}$. Recall, from our discussion in the SMTC case, that finding how much the window should be shrunken is difficult. It is even more difficult to find $\Omega^{y \to k}$ when mpc^y can be assigned any value. For this reason, we will again find instead a lower bound on how much the window should be shrunken. Clearly this offers an upper bound on the window, and thus it is safe. Let $\Omega^{y \to k}$ denote a lower bound on the amount that the window of node y should be shrunken due to the address counter evolution at the end of the $Q^{y \to k}$. Thus, we have:

$$\Omega^{y \to k}(t) = \begin{cases} 0, & \text{if } y = k \\ T_{MS} \times n_{slots}^{y \to k}(t) + T_{PR} + \Omega^{next(y) \to k}(t), & \text{if } y \neq k \end{cases} \quad (21)$$

The intuition behind (21) is similar to the intuition provided for $\Omega^{y \to k}$ in the SMTC case. If $y = k$, then we are looking at the skipped slots on the node where the message from S_i^k is assigned; that is, the window should not be shrunken at all, and hence $\Omega^{y \to k}$ should be zero. Otherwise y and k are different nodes. Then, it matters if node y transmitted a message at the end of the window $Q^{y \to k}$. If it did, then the window of node y should finish earlier than the window of node *next*(y).

In order to know if a message was transmitted on node y at the end of the window $Q^{y \to k}$, we compute the length of the queue of output messages at node y. We also need to know how many messages were transmitted ($n_{slots}^{y \to k}(t)$ denotes that). Finding if a message is transmitted is hard however, so instead we will use a lower bound. If the lower bound on the queue length is 1 or greater, then we know that a message was transmitted. $LBql^{y \to k}$ denotes this lower bound. If we know $LBql^{y \to k}$, then we can compute $n_{slots}^{y \to k}$ as follows:

$$n_{slots}^{y \to k}(t) = \min \left\{ mpc^y, \max \left\{ 0, LBql^{y \to k}(t) \right\} \right\} \quad (22)$$

A lower bound on the queue length must be 0 or more, hence the term $max\ \{0, LBql^{y \to k}(t)\}$ in (22). It represents another lower bound on the queue length. If, however, this would be greater than mpc^y, then $n_{slots}^{y \to k} = mpc^k$, and thus this is the maximum number of messages that node y can transmit in the last TDMA cycle.

We will now focus on computing $LBql^{y \to k}$. Let ql^y denote the length of the output queue of node y at time $L^{y \to k}(t)$ after the message from S_i^k was put in the output queue. $L^{y \to k}(t)$ is given by:

$$L^{y \to k}(t) = \max \left\{ 0, t - \left(\Omega^{next(y) \to k}(t) + mpc^y \times T_{MS} + T_{PR} \right) \right\} \quad (23)$$

As a message from S_i^k was in the queue at the end of the time window Q_i^k, clearly it must have been in the queue earlier. Hence, we know that $1 \leq ql^k$. Since the

queue length of node k depends on the number of arrived messages and on the number of transmitted messages, we obtain (24):

$$ql^k \le \sum_{\forall S_j^k \, on \, N^k} \left\lceil \frac{L^{y \to k}(t)}{T_j^k} \right\rceil - ntransmitted^k \qquad (24)$$

where $ntransmitted^k$ denotes the number of messages transmitted during the time window of length $L^{y \to k}$. Using a similar reasoning we get:

$$ql^y \ge \sum_{\forall S_j^y \, on \, N^y} \left\lfloor \frac{L^{y \to k}(t)}{T_j^y} \right\rfloor - ntransmitted^y \qquad (25)$$

Observe that (24) and (25) offer a lower/upper bound on the output queue length, and that they refer to the queue length at different nodes.

Consider those TDMA cycles such that node y transmitted at least one message during the time interval of length $L^{y \to k}$. Let $nTDMArounds^y$ denote the number of those TDMA cycles. We know that the network is fair, in the sense that in a time interval two different nodes receive almost the same number of TDMA cycles. It follows that the difference between the number of TDMA cycles received by any two nodes is at most one:

$$nTDMAcycles^y \le nTDMAcycles^k + 1 \qquad (26)$$

Since node k used all its messages in all its time slots during the window of length Q_i^k, it also used all its time slots in the window of length $L^{y \to k}(t)$. This implies that all its TDMA cycles transmitted mpc^k messages. Therefore:

$$nTDMAcycles^k \le \left\lceil \frac{ntransmitted^k}{mpc^k} \right\rceil \qquad (27)$$

On node y, we do not know whether slots are skipped or not and how many slots are skipped. We do know however that every TDMA cycle can transmit at most mpc^y messages. Hence, we have:

$$ntransmitted^y \le nTDMAcycles^y \times mpc^y \qquad (28)$$

Combining (26), (28) and (27) yields:

$$ntransmitted^y \le \left(\left\lceil \frac{ntransmitted^k}{mpc^k} \right\rceil + 1 \right) \times mpc^y \qquad (29)$$

We have already seen that $1 \le ql^k$. Combining it with (24), (25) and (29) leads to (30).

$$LBql^{y \to k}(t) = \sum_{\forall S_j^y \, on \, N^y} \left\lfloor \frac{L^{y \to k}(t)}{T_j^y} \right\rfloor - \left(\left\lceil \frac{\left(\sum_{\forall S_j^k \, on \, N^k} \left\lceil \frac{L^{y \to k}(t)}{T_j^k} \right\rceil \right) - 1}{mpc^k} \right\rceil + 1 \right) \times mpc^y \qquad (30)$$

The expression for $LBql^{y \to k}$ in (30) can be used in (22) and then in (21) to obtain the value of $\Omega^{y \to k}$. We can also see that (30) is a generalisation of (14).

4.2. Heuristic

Having analysed the behaviour of MMTC, we will now focus on the problem of assigning *mpc* values to nodes.

It would be tempting to use a scheme that assigns mpc^k to be proportional to $\sum_{\forall S_j^k \text{ on } N^k} \lceil T_{MS} / T_j^k \rceil$ as was done in the normalised proportional allocation scheme used in timed token networks [8]. However, applying such an algorithm in TDMA/SS is non-trivial for two reasons. Firstly, in TDMA/SS, the TDMA cycle time may vary at run-time, because the number of skipped slots may be different in different TDMA cycles. Secondly, if $D_i^k \ne T_i^k$, then it is not obvious how to compute the utilisation of a message stream, and therefore to compute the utilisation of a node.

The idea we use was demonstrated by the example outlined in Figure 3: nodes having message streams that miss a deadline should receive a larger *mpc*. Thus, a simple algorithm to apply this concept is sketched below (Algorithm 1).

Algorithm 1: Assigning *mpc* to nodes.

```
1.  for all nodes k: mpc^k ← 1 end for
2.  while ( Σ_{k=1}^{n} mpc^k ≤ ⌈ TMIN / T_MS ⌉ ) do begin
3.      for all nodes k
4.          for all messages streams S_i^k at node k:
5.              Compute all Q_i^k using (19)
6.          end for
7.      end for
8.      if all messages meet their deadlines
9.          return SUCCESS
10.     else
11.         for all nodes k such that there is a messages
              stream on node k that misses a deadlines
12.             mpc^k ← mpc^k + 1
13.         end for
14.     end if
15. end while
16. return FAILURE
```

Observe that since there is a node with mpc^k that increases in each iteration, there will be at most $\lceil TMIN / T_{MS} \rceil$ iterations of the lines 2-15. Considering that (19) is used to compute Q_i^k has an upper bound of $max\{\forall S_i^k \text{ on } N^k \; T_i^k\}$ iterations, the algorithm to assign *mpc*s to nodes has a low computational complexity.

Another advantage of our algorithm is that if the workload of a node is not known, the algorithm can attempt to find it anyway by replacing line 8 with detecting deadline misses. When a deadline is missed, then we execute lines 11-13 and continue operation for some time (typically some multiples of

the maximum period). At that point, deadlines misses are detected and iteration 2-15 starts again. After a long time of no deadline misses, a node should decrease its *mpc*, if it is greater than 1. With such an application, a node does not need global knowledge (such as the normalized allocation scheme in [8] does), but only local knowledge is needed.

5. Numerical Example

We have developed a tool (called TDMA analyser) to compare the real queuing times q_i^k with the upper bound on the queuing times Q_i^k. In the test scenarios we have run, the analysis is often tight; that is, $q_i^k = Q_i^k$. Nevertheless, there are some message streams on which the analysis is not tight. We will now look at it to understand the reason for this small level of pessimism, to give an idea of how large it can be, and to illustrate how the calculations are made. Consider the network example given by Figure 4.

We calculated the *mpc*s by Algorithm 1, with resulting values as illustrated in Figure 4. This assignment of *mpc*s is good since all deadlines are met and our analysis claims (by calculating Q_i^k) that all deadlines are met. Furthermore, there is no other assignment of *mpcs* with a lower T_{TDMA}.

Looking now at the behaviour of the protocol after time 0, we can see that the queuing delay $q_2^3 = 15.4$. This is less than the calculated upper bound on the queuing delay $Q_2^3 = 19.4$. To understand this, look at node 4 at time instant 14.4. A message of message stream S_1^4 arrives at time instant 14.4. So, without using our analysis based on Ω, we would have concluded that

$$net = \left(4, \left\{N^1, N^2, N^3, N^4\right\}, 1, 1/5\right)$$

$$\begin{cases} N^1 = \left(3, \left\{S_1^1, S_2^1, S_3^1\right\}, 2\right) \\ \quad T_1^1 = D_1^1 = 8.0 \\ \quad T_2^1 = D_2^1 = 10.0 \\ \quad T_3^1 = D_3^1 = 25.0 \end{cases} \quad \begin{cases} N^2 = \left(4, \left\{S_1^1, S_2^1, S_3^1\right\}, 2\right) \\ \quad T_1^2 = D_1^2 = 9.0 \\ \quad T_2^2 = D_2^2 = 15.0 \\ \quad T_3^2 = D_3^2 = 20.0 \\ \quad T_4^2 = D_4^2 = 30.0 \end{cases}$$

$$\begin{cases} N^3 = \left(2, \left\{S_1^3, S_2^3\right\}, 1\right) \\ \quad T_1^3 = D_1^3 = 10.0 \\ \quad T_1^3 = D_1^3 = 27.0 \end{cases} \quad \begin{cases} N^4 = \left(1, \left\{S_1^4\right\}, 1\right) \\ \quad T_1^4 = D_1^4 = 15.0 \end{cases}$$

Figure 4. MMTC message streams.

the message that arrived at time 14.4 would be transmitted before time 15.4, and hence has caused interference. However, we can see that, at time instant 11.8, node 4 has `address_counter` with value 4. Moreover, at that time instant its output queue is empty, and after that, node 4 will not have `address_counter`=4 before time instant 15.4. Hence, the message that arrived at time 14.4 does not cause interference on S_2^3. A tight analysis must recognise this and observe that node 4 skips a slot at time 11.8.

Let us now turn our attention to see how our analysis deals with this. Our analysis performs the following iterations of Q_2^3: 0, 5.8, 12.6, 18.4, finally converging to 19.4. The real queuing time $q_2^3 = 15.4$ is never considered by the analysis, but if it would be considered, it would be deemed to be too small. In order to understand this, insert $Q_2^3 = 15.4$ in the right hand side of (19). We need to compute $nss^{4\to 3}(15.4,2)$ from (20).

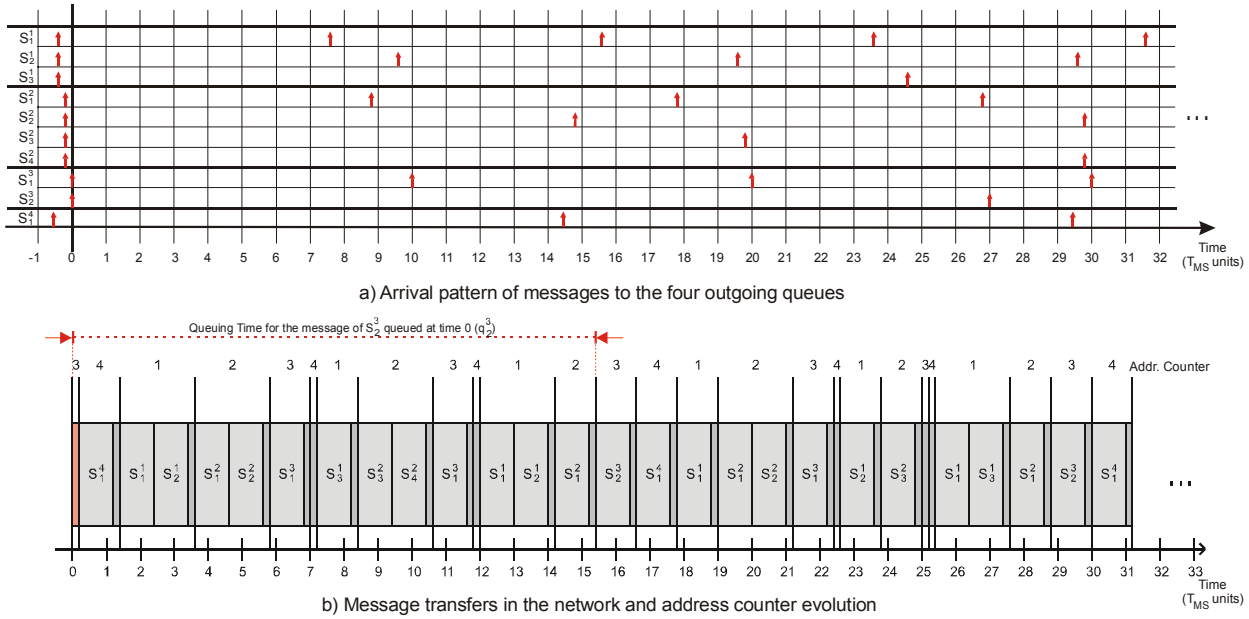

Figure 5. Arrival times and schedule of MMTC network.

This requires computing $\Omega^{2\to3}(15.4)$ from (21), and hence we need to compute $n_{slots}^{2\to3}(15.4)$ from (22). We obtain $n_{slots}^{2\to3}(15.4) = 0$, but in fact one message was transmitted during the time interval [*14.2*, *15.4*), and thus a more accurate analysis of the window should have computed $\Omega^{2\to3}(15.4)=1 \times T_{MS} + T_{PR}=1.2$. With our analysis (21), we obtain $\Omega^{2\to3}(15.4)=T_{PR}=0.2$. Repeatedly applying (21) gives us $\Omega^{1\to3}(15.4)=2 \times T_{PR}=0.4$, and $\Omega^{4\to3}(15.4)=3 \times T_{PR}=0.6$.

From (12), we obtain: $\Phi^{4\to3} = 3 \times T_{PR} = 0.6$. We are now ready to apply (20) to compute the number of skipped slots at node 4. We obtain:

$$nss^{4\to3}(15.4,2) = \max\left\{0, \left\lfloor \frac{\left\lceil \frac{15.4}{10} \right\rceil}{1} \right\rfloor \times 1 - \left(1 + \left\lfloor \frac{15.4 + 0.6 - 0.6}{15} \right\rfloor\right)\right\} = 0$$

Hence, our analysis does not detect the skipped slot on node 4, and this is the source of the pessimism illustrated in Figure 5. This example (Figure 4) shows that our analysis is not exact; but we err on the safe side. We have chosen this example because of all message streams we have simulated, this is the one with most pessimism, and still it is reasonably small.

6. Discussion and Related Work

TDMA/SS has the following advantages. First, it is not dependent on bit-level synchronisation, and hence the speed can be quite high (unlike the binary countdown protocol [9] used in CAN). Second, TDMA/SS does not require sensing-while-transmitting. Third, TDMA/SS relies on nodes that are equipped with a real-time clock, but it does not depend on them being synchronised; nodes only need to listen for the protocol slot of length T_{PR} to update the `address_counter`. (If nodes remain silent for a long time there may be a need to transmit a dummy message in order to keep synchronisation; if this occurs for periodic traffic then it must have been that the utilisation was low and hence this overhead of dummy messages should not be a problem). Fourth, TDMA/SS can (if we use $\Omega^{y\to k} = 0$) be used to schedule sporadic [9] message streams. Fifth, TDMA/SS is energy-efficient because it is collision-free and the network-controller only needs to listen in the beginning of a new slot (to determine whether the slot was a message slot or a protocol slot). Sixth, TDMA/SS is resilient to crashes if nodes are fail-silent. (One way to implement TDMA/SS is that a node transmitting a protocol slot keeps silent for T_{PR} time units. Then, if a node y crashes, this idle time will cause, `address_counter` to become *next*(y) after T_{PR} time units, and hence the operation of the other nodes are unaffected.)

As already mentioned, a TDMA/SS-like protocol was studied in [5] but it had the drawbacks of (i) lacking an accurate calculation of Ω, (ii) lacking the opportunity to transmit multiple messages per TDMA cycle, and (iii) assuming FIFO scheduling on each node.

The TDMA/SS protocol has similarities to the ARINC 629 protocol [10] in that ARINC 629 is a TDMA protocol which does not need synchronised clocks. Nodes are given time slots in a pre-specified order; they have a terminal gap (TG) specifying an idle between nodes (similar to our T_{PR}) and they permit slot skipping. Unfortunately, their analysis is not accurate in the sense that they do neither take into account effects like the Φ and the Ω, nor the local scheduling of output queues.

Scheduling messages in TDMA without slot skipping [1-3] is well studied but, as we have already mentioned, they may require long TDMA cycles. Usually they create schedules before run-time. However, one recently proposed protocol [11] creates the schedule at run-time in a distributed fashion. First, it selects periods (shorter than required) to make sure that periods are harmonic. Then, at run-time, when a collision is detected, a winner of the colliding nodes is elected. The winning node will transmit and it is assigned an offset so future collisions cannot occur. Such an approach is efficient in the sense that no time is wasted on protocol slots. However, synchronised clocks are required, and sporadic message streams cannot be efficiently scheduled.

The timed token protocol is similar to TDMA/SS, and it has been used in FDDI rings and IEEE 802.5. Schedulability analysis techniques and algorithms to assign H_k (similar to our mpc^k) have been developed [8, 12, 13]. These protocols differ from TDMA/SS in that they explicitly pass a token while TDMA/SS does not. Timed token networks have a target token circulation time. This is similar to our T_{TDMA}, but there is one important difference though. If the token circulates faster in one circulation, then this time can be used on a node to transmit soft real-time messages (this is called asynchronous). In TDMA/SS however, the `address_counter` will actually change faster, and hence there will be more capacity for hard real-time traffic. Hence, there are hard real-time message streams that can be scheduled with TDMA/SS but that cannot be scheduled with the timed token protocol. The analysis of timed token protocols performed in holistic scheduling [14, 15] addresses a problem similar to ours (the S_p in [14] is equivalent to our mpc^p;

in [15] mpc^k is more restricted, it is assumed to be 1, and [15] uses EDF to schedule messages from the same node). However, neither [14] nor [15] take the Φ and $\Omega^{y \to k}$ into account or something similar (issues due the fact that this is a distributed system).

Real-time scheduling on IEEE 802.5 networks were studied in [16]. It uses explicitly message passing where a token must circulate and nodes announce their priority before transmitting. That is unlike TDMA/SS which only prioritise messages on each node.

Implicit EDF is a TDMA MAC protocol recently proposed [17]. It assumes that all nodes know all messages streams in the system. Every node computes the earliest deadline of all those message streams and hence at most one message is transmitted at every time. Such a protocol offers faster response to urgent events than TDMA/SS does. However, their protocol has three drawbacks. First, the protocol requires knowledge of all message streams of other nodes. Second, they depend on synchronised clocks. Third, sporadic messages cannot be efficiently scheduled.

7. Conclusions and Future Work

We conclude that the TDMA/SS protocol has attractive real-time and energy-efficient properties suited for real-time applications. For future work, we consider nodes that are non-work-conserving; that is, they are idle although they have non-empty output queue. This can make the `address_counter` change faster and it is necessary when arbitrating if some message streams have very fine-grained deadlines. We also would like to explore techniques that permit a node to sleep for an extended period and still maintain consistent `address_counter` when it wakes up. This is important to make the protocol not only energy-efficient but also to achieve low power consumption.

8. References

[1] H. Kopetz and G. Grunsteidl, "TTP-a protocol for fault-tolerant real-time systems", IEEE Computer, vol. 27(1), pp. 14-24, 1994.

[2] L. Dong, R. Melhem, and D. Mossé, "Scheduling Algorithms for Dynamic Message Streams with Distance Constraints in TDMA protocol", in proceedings of the 12th Euromicro Conference on Real-Time Systems (ECRTS'00), pp. 239-246, 2000.

[3] C.-C. Han, K.-J. Lin, and C.-J. Hou, "Distance-constrained scheduling and its applications to real-time systems", IEEE Transactions on Computers, vol. 45(7), pp. 814 -826, 1996.

[4] IPUO, "The P-NET Standard": International P-NET User Organisation, 1994.

[5] E. Tovar, F. Vasques, and A. Burns, "Communication Response Time in P-NET Networks: Worst-Case Analysis Considering the Actual Token Utilisation", Real-Time Systems Journal, Kluwer Academic Publishers, vol. 22(3), pp. 229-249, 2002.

[6] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", Journal of the ACM (JACM), vol. 20(1), pp. 46-61, 1973.

[7] M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System", The Computer Journal, British Computer Society, vol. 29(5), pp. 390-395, 1986.

[8] G. Agrawal, "Guaranteeing Synchronous Message Deadlines with the Timed Token Medium Access Control Protocol", IEEE Transactions on Computers, vol. 43(3), pp. 327 - 339, 1994.

[9] A. Mok, "Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment", PhD thesis, Massachusetts Institute of Technology, Cambridge, Mass., 1983. Available online at http://www.lcs.mit.edu/publications/specpub.php?id=865.

[10] N. Audsley and A. Grigg, "Timing analysis of the ARINC 629 databus for real-time application", Microprocessors and Microsystems, vol. 21, pp. 55-61, 1997.

[11] T. W. Carley, M. A. Ba, R. Barua, and D. B. Stewart, "Contention-Free Periodic Message Scheduler Medium Access Control in Wireless Sensor / Actuator Networks", in proceedings of the Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS'03), pp. 298, 2003.

[12] S. Zhang and A. Burns, "An Optimal Synchronous Bandwidth Allocation Scheme for Guaranteeing Synchronous Message Deadlines with the Timed-Token MAC Protocol", IEEE/ACM Transactions on Networking (TON), vol. 3(6), pp. 729 - 741, 1995.

[13] N. Malcolm and W. Zhao, "The timed-token protocol for real-time communications", IEEE Computer, vol. 27(1), pp. 35-41, 1994.

[14] K. Tindell, "Analysis of hard real-time communications", University of York Dept. of Computer Science, Heslington, York, England, Technical report YCS-94-222, 1994. Available online at ftp://ftp.cs.york.ac.uk/reports/YCS-94-222.ps.Z.

[15] M. Spuri, "Holistic Analysis for Deadline Scheduled Real-Time Distributed Systems", INRIA, Technical Report RR-2873, April 1996. Available online at http://www-rocq.inria.fr/reflecs/research_reports/RR-2873.pdf.

[16] J. K. Strosnider, T. Marchok, and J. Lehoczky, "Advanced Real-time Scheduling Using the IEEE 802.5 Token Ring", in proceedings of the 9th IEEE Real-Time Systems Symposium (RTSS'88), Huntsville, Alabama, USA, pp. 42-52, 1988.

[17] M. Caccamo and L. Y. Zhang, "An Implicit Prioritized Access Protocol for Wireless Sensor Networks", in proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02), Austin, Texas, pp. 39-48, 2002.