Technical Report

Analysing TDMA with Slot Skipping

Björn Andersson

Nuno Pereira

Eduardo Tovar

HURRAY-TR-081103

Version: 0

Date: 10-18-2008

Analysing TDMA with Slot Skipping

Björn Andersson, Nuno Pereira, Member, Eduardo Tovar

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:

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

Abstract

Distributed real-time systems, such as factory automation systems, require that computer nodes communicate with a known and low bound on the communication delay. This can be achieved with traditional Time-Division-Multiple-Access (TDMA). But improved flexibility and simpler upgrades are possible through the use of TDMA with Slot-Skipping (TDMA/SS), meaning that a slot is skipped whenever it is not used and consequently the slot after the skipped slot starts earlier. We propose a schedulability analysis for TDMA/SS. We assume knowledge of all message streams in the system, and that each node schedules messages in its output queue according to Deadline Monotonic. Firstly, we present a non-exact (but fast) analysis and then, at the cost of computation time, we also present an algorithm that computes exact queuing times.

Analysing TDMA with Slot Skipping

Abstract

Distributed real-time systems, such as factory automation systems, require that computer nodes communicate with a known and low bound on the communication delay. This can be achieved with traditional Time-Division-Multiple-Access (TDMA). But improved flexibility and simpler upgrades are possible through the use of TDMA with Slot-Skipping (TDMA/SS), meaning that a slot is skipped whenever it is not used and consequently the slot after the skipped slot starts earlier. We propose a schedulability analysis for TDMA/SS. We assume knowledge of all message streams in the system, and that each node schedules messages in its output queue according to Deadline Monotonic. Firstly, we present a non-exact (but fast) analysis and then, at the cost of computation time, we also present an algorithm that computes exact queuing times.

1. Introduction

A fundamental problem in distributed real-time systems is the sharing of a communication medium between message streams on different nodes such that real-time requirements are satisfied. Time division multiple access (TDMA) communication protocols solve this problem 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 of real-time message

streams. This is the case for analysis over time-triggered protocols such as TTP [1]. It is also the case of work addressing distance constraints (maximum timing interval between two adjacent messages of the same message stream) as an additional temporal restriction [2, 3]. Unfortunately, an unused slot is wasted and cannot be used for other hard real-time traffic. In order to meet all deadlines, it may be necessary that a message stream with periodic messages uses a specific time slot in a TDMA cycle only in a few cycles, while in most cycles that time slot is not used, hence wasted. One way to overcome this waste is to have a larger TDMA cycle serving several messages 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. This model is applicable to P-NET, a commercial-off-the-shelf (COTS) technology [4], defined in an International Fieldbus Standard. For this generic class of TDMA networks, a schedulability analysis that takes slot skipping into account is still missing.

In this paper we present a schedulability analysis for TDMA networks with slot skipping (TDMA/SS). We assume that all message streams are known, and that each node schedules messages in its output queue according to deadline monotonic (DM). We present two complementing analyses, (i) an analysis which is fast but not exact and (ii) an analysis which is exact but with a larger time-complexity.

As already mentioned, the analysis of TDMA/SS is applicable to COTS technology, in particular the P-NET standard [4]. This paper advances the state-of-art in two ways. First, both of our analyses are tighter than any other previous analysis on TDMA networks that skips slots [5]. Second, we also consider the case where a node can be assigned a fixed number of slots, whereas previous work [5] 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 and provides an understanding of key ideas for analyzing TDMA/SS. Section 3 presents a non-exact analysis; it presents an algorithm for finding an upper bound on the queuing delay of

a message. Section 4 presents an exact analysis; it presents an algorithm for finding the maximum queuing delay that a message can experience. Section 5 compares our approach to other approaches in real-time communications, and finally, in Section 6, conclusions are drawn.

2. Preliminaries

In this section, the problem of analysing TDMA/SS networks is introduced. We start by defining the network and message models and proceed to present a simple network example that allows us to better illustrate our model. This example also allows for demonstrating the reason why the schedulability analysis of TDMA/SS networks is not trivial. We then discuss the problem of finding worst-case queuing times, introducing concepts that will help us in Section 3 and Section 4, where the new schedulability analysis techniques are presented.

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, such as the P-NET standard [4], 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 with each node k (k ranging from 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:

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

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 deadline monotonic (DM) scheduling 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 $r_i^k \leq D_i^k$ then we say that S_i^k meets its deadlines. We are interested in finding out whether all messages meet their deadlines. Hence, we will find Q_i^k , an upper bound on 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 $R_i^k \leq D_i^k$ then we say that S_i^k is deemed to meet its deadlines according to our analysis technique. Our analysis assumes that $D_i^k \leq T_i^k$. 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}.

When describing the TDMA/SS protocol and related time analysis, some shorthand notations are useful. The next and the previous nodes are given by circularly incrementing the current address k:

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

Because we use DM, the notations $hp^k(S_i^k)$ and $lp^k(S_i^k)$ are useful to denote the subset of message streams on node k with higher or lower priority than S_i^k respectively, and are defined as:

$$\begin{aligned} hp^k(S_i^k) &= \left\{ S_j^k : D_j^k < D_i^k \right\} \cup \left\{ S_j^k : \left(D_j^k = D_i^k \right) \wedge \left(j < i \right) \right\} \\ lp^k(S_i^k) &= \left\{ S_j^k : D_j^k > D_i^k \right\} \cup \left\{ S_j^k : \left(D_j^k = D_i^k \right) \wedge \left(j > i \right) \right\} \end{aligned} \tag{4}$$

We will now describe the operation of the network protocol being used. During the operation of the protocol, all nodes maintain at all time a variable – `address_counter` – that keeps track of the node

holding the right to transmit. `address_counter` has the same value on all nodes, and thus in the discussion we treat it as a single 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 $0 \leq x < mpc^k$ messages, then only those *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 the network and message models, consider the following network:

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

$$\begin{cases} N^1 = \left(4, \left\{S_1^1, S_2^1, S_3^1, S_4^1\right\}, 1\right) \\ T_1^1 = D_1^1 = 7.0 \\ T_2^1 = D_2^1 = 12.0 \\ T_3^1 = D_3^1 = 13.4 \\ T_4^1 = D_4^1 = 21.0 \end{cases} \quad \begin{cases} N^2 = \left(1, \left\{S_1^2\right\}, 1\right) \\ T_1^2 = D_1^2 = 5.3 \end{cases} \quad \begin{cases} N^3 = \left(1, \left\{S_1^3\right\}, 1\right) \\ T_1^3 = D_1^3 = 7.0 \end{cases}$$

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 assume:

i) a message from S_4^1 arrives marginally before time 0;

ii) the `address_counter` changes from 3 to 1 at time 0;

iii) and messages from S_1^1, S_2^1 and S_3^1 arrive at time 0.

We also assume that a message is only allowed to be transmitted by node *k*, if and only if it has been queued before `address_counter` changes to the value *k*. (This assumption is true in P-NET, a commercial-off-the-shelf (COTS) technology [4].) As a result, for the exemplified scenario, the messages

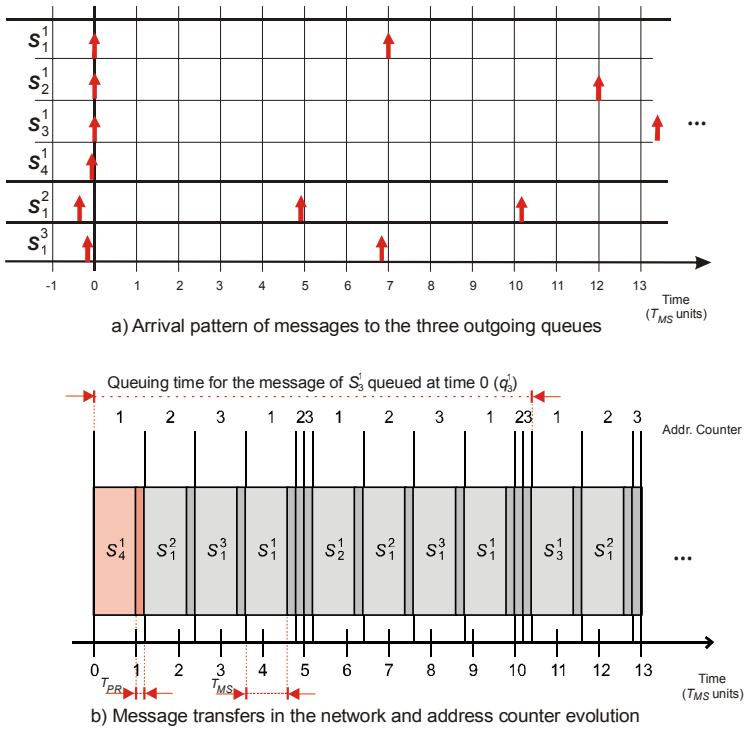

Figure 2. Arrival times and schedule of the example network scenario.

from S_1^1, S_2^1 and S_3^1 are not transmitted at time 0. Instead, a message from S_4^1, 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.

Observe for time $t > 0$, 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_3^1 that was placed in the output queue at time 0. This message is queued during [0,10.4) and hence q_3^1 is 10.4. The message of S_3^1 is blocked during the time interval [0,3.6) because some messages, a lower priority message S_4^1 and other messages S_1^2 and S_1^3, cause S_3^1 to be queued although it has higher priority. The message of S_3^1 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 10.2. At this time instant, a message from S_3^1 (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_3^1, transmitted at time instant 10.4. In general, when finding the queuing time of a message from message stream S_i^k, we clearly need to find out if another node y skipped a slot (and if so how many slots) during this queuing time. In order to find how many slots were used and how many were skipped on node y, it is required that we consider a time window on node y. Finding this time window on node y is a major challenge that we will deal with in this paper.

2.3. Finding Worst-Case Queuing Times

To characterize the critical instant under DM scheduled output queues in TDMA/SS, it is tempting to reuse the condition for the critical instant used in DM on a single processor; just that blocking needs to be considered. One might believe that the critical instant of a message of message stream S_i^k occurs when it arrives simultaneously with all other message streams, except that one message of a message stream with the lowest priority on the same node as S_i^k arrives marginally before and this message is transmitted causing blocking. However, consider the example shown in Figure 2 and analyze the queuing delay of the message from S_3^1 that arrived at time 0. It can be seen that a message on node 3 (in this case message from message stream S_1^3) can arrive at time $-T_{PR}$ and still cause as much interference on S_3^1 as if S_1^3 would have arrived at time 0. This is due to the way `address_counter` is incremented and it has no parallel in DM on a single processor.

However, we can use a shifting argument similar to the one used in [6] to show how early should a message arrive at on node y to cause Q_i^k to be maximized. 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 this message 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*)), node *prev*(*prev*(*prev*(*k*))), and so on. Let $\Phi^{y\to k}$ denote the amount of time which messages from nodes y should arrive earlier in order to delay the queuing times of the messages in node k the most. $\Phi^{y\to k}$ is given as:

$$\Phi^{y\to k} = \begin{cases} 0 & if\ y = k \\ T_{PR} + \Phi^{next(y)\to k} & if\ y \neq k \end{cases} \tag{5}$$

We are now in position to say that a critical instant of an arbitrary message stream S_i^k is at time t under the following conditions:

1. S_i^k releases a message at time t;
2. if $lp^k(S_i^k)\neq\varnothing$ then a message stream in $lp^k(S_i^k)$ released a message infinitesimally before time t;
3. all other message streams in each node y ($\forall y : 1 \le y \le n\ y\neq k$) are synchronously activated $\Phi^{y\to k}$ time units before t (where $\Phi^{y\to k}$ is given by (5)).

As in [7], for non-premptive uniprocessor DM scheduling, and [8] for the CAN bus, the maximum queuing time for message stream S_i^k is found within the length Lbp^k of the busy period (for convenience, we do not consider the concept of level-i busy period; note that our calculated busy periods will always be no less than a level-i busy period). Specifically, we can find the maximum queuing time for a message stream S_i^k by analysing the schedule patterns resulting from the different activation times $a \in A_i^k$:

$$A_i^k = \bigcup_{j=1}^{ns^k} \left\{ c \times T_j^k : c \in \mathbb{N}_0 \right\} \cap \left[0, Lbp^k \right) \tag{6}$$

($\mathbb{N}_0$ represents the set of non-negative integers) and the queuing time is given by:

$$q_i^k = \max_{\forall a \in A_i^k} \left\{ q_i^k(a) - a \right\} \tag{7}$$

where $q_i^k(a)$ is the time of the start of the transmission of the message from S_i^k with arrival time a.

Now, we are left with the problem of finding the length Lbp^k of the busy period. One simple option is to consider that $Lbp^k = lcm$ ($\forall T_j^k$ on $node^k$). For some sets of streams (for example, a set of streams with periods that are prime numbers) this will be very long, resulting in a great amount of activation times to

be tested. In Section 4, we will exploit the idea of using an algorithm that simulates TDMA/SS networks over time to compute the length of the busy period. This solution allows finding the exact length of the busy period, at the cost of higher time complexity.

In our non-exact analysis (Section 3), instead of trying to find the maximum queuing time of a message within the length Lbp^k of the busy period, we adapt a simple technique proposed in [8] for our setting. It results in a more pessimistic analysis, but greatly reduces the complexity by avoiding the computation of multiple queuing times for the same message stream.

3. Non-Exact Analysis

In this section we derive an upper bound on the queuing time for message streams. We will first (in Section 3.1) present the schedulability analysis assuming a lower bound on the number of skipped slots is known and then (in Section 3.2) compute such a lower bound.

3.1. Queuing Time Equation

Response time equations [9] for static-priority scheduling on a uniprocessor can be extended to the problem of finding the queuing delay in communication networks. This is carried-out in many analyses such as [10-12]. Inspired by this, and the technique proposed in [8] to avoid situations where previous analysis would be optimistic, we can compute Q_i^k of a message stream considering that nodes never skip slots by reasoning as follows. Clearly, Q_i^k depends on the number of time slots needed by message streams of higher priority than S_i^k. For this reason, let us define

$$slots_i^k\left(Q_i^k\right) = \sum_{S_j^k \in hp^k\left(S_i^k\right)} \left\lceil \frac{Q_i^k}{T_j^k} \right\rceil .$$

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, Q_i^k can be computed as:

$$Q_i^k = \max\left\{B_i^k, T_{MS} + T_{PR}\right\} + \left\lfloor \frac{slots_i^k\left(Q_i^k\right)}{mpc^k} \right\rfloor \times T_{TDMA} + \left(slots_i^k\left(Q_i^k\right) \bmod mpc^k\right) \times T_{MS} \quad (8)$$

where T_{TDMA} corresponds to the TDMA cycle duration when no slots are skipped. The term $\max\{B_i^k, T_{MS} + T_{PR}\}$ in (8) is needed to account for blocking due to the non-preemptive nature of message transmissions and the TDMA mechanism.

Equation (8) can be refined to include the effect of slot skipping. This gives us:

$$Q_i^k = \max\left\{B_i^k, T_{MS} + T_{PR}\right\} + \left\lfloor \frac{slots_i^k\left(Q_i^k\right)}{mpc^k} \right\rfloor \times T_{TDMA} + \left(slots_i^k\left(Q_i^k\right) \bmod mpc^k\right) \times T_{MS} - \left\lfloor \sum_{y=1, y\neq k}^{n} nss^{y\to k}\left(Q_i^k, i\right) \right\rfloor \times T_{MS} \quad (9)$$

where $nss^{y\to k}(Q_i^k, i)$ denotes a lower bound on the number of skipped slots on node y when the queuing time of S_i^k is at most Q_i^k. The term T_{MS} in (9) represents the amount of time saved when a slot is skipped. In the next subsection we will provide the reasoning for the analysis on the number of skipped slots.

We must now define T_{TDMA} and the blocking factor B_i^k. The term T_{TDMA} can be interpreted as the maximum time interval that can elapse between two consecutive accesses to the network by one particular node. In our network model, T_{TDMA} and B_i^k are given by:

$$T_{TDMA} = \left(\sum_{l=1}^{n} mpc^l\right) \times T_{MS} + n \times T_{PR} \quad (10)$$

$$B_i^k = \left[\left(\sum_{l=1, l\neq k}^{n} mpc^l\right) + \min\left\{mpc^k, \left|lp^k\left(S_i^k\right)\right|\right\}\right] \times T_{MS} + n \times T_{PR} \quad (11)$$

3.2. Determining the Number of Skipped Slots

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 exactly 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 of messages must be non-negative, we can state that:

$$LBnumber\ of\ unused\ slots\ on\ N^y = \max\{0, LBnumber\ of\ slots\ that\ were\ available\ to\ node\ y - UBnumber\ of\ slots\ that\ was\ used\ by\ node\ y\}$$

Since each TDMA cycle makes available mpc^k slots on node k, we can compute the number of TDMA cycles during Q_i^k. We also know that every time node y has the right to transmit, node y has mpc^y slots available. This gives us:

$$LBnumber\ of\ slots\ that\ were\ available\ to\ node\ y = \left\lfloor \frac{slots_i^k\left(Q_i^k\right)}{mpc^k} \right\rfloor \times mpc^y$$

Recall from the discussion in Figure 2 that when considering the effect of unused slots at another node y on S_i^k, we should not consider the time window of the queuing time for S_i^k, instead we must consider a time window that starts earlier and ends earlier. Recall from Section 2 that $\Phi^{y\to k}$ denotes how much earlier it starts. We let $X^{y\to k}$ denote how much earlier it ends. It is difficult however to compute $X^{y\to k}$, so instead we find $\Omega^{y\to k}$ such that $\Omega^{y\to k} \le X^{y\to k}$. Based on this, we can state that:

$$UBnumber\ of\ slots\ that\ was\ used\ by\ node\ y = ns^y + \sum_{\forall S_j^y\ on\ N^y} \left\lfloor \frac{Q_i^k + \Phi^{y\to k} - \Omega^{y\to k}\left(Q_i^k\right)}{T_j^y} \right\rfloor$$

Combining these equations gives us:

$$nss^{y\to k}\left(Q_i^k, i\right) = \max\left\{0, \left\lfloor \frac{slots_i^k\left(Q_i^k\right)}{mpc^k} \right\rfloor \times mpc^y - \left(ns^y + \sum_{\forall S_j^y\ on\ N^y} \left\lfloor \frac{Q_i^k + \Phi^{y\to k} - \Omega^{y\to k}\left(Q_i^k\right)}{T_j^y} \right\rfloor \right)\right\} \quad (12)$$

In order to calculate $nss^{y\to k}(Q_i^k, i)$ using (12) we need to calculate $\Omega^{y\to k}$. To understand the notion of $\Omega^{y\to k}$, let us consider the special case $\Omega^{prev(k)\to k}$. Let u denote the time of the transmission of the message from S_i^k. Also, let v denote the latest time such that (i) at time v it holds that `address_counter` transitions to $prev(k)$ and (ii) $v<u$. It is clear that any messages that arrive on node $prev(k)$ after time v cannot affect the queuing time of the message from S_i^k. Because of that we have $\Omega^{prev(k)\to k}=u-v$. Note that $u-v$ is the time that node $prev(k)$ had `address_counter` = $prev(k)$ and hence $\Omega^{prev(k)\to k}$ depends on how many messages node $prev(k)$ transmitted. Therefore, we have:

$$\Omega^{prev(k)\to k}\left(Q_i^k\right)=T_{MS}\times n_{slots}^{prev(k)\to k}\left(Q_i^k\right)+T_{PR}$$

where $n_{slots}^{prev(k)\to k}$ is a lower bound on the number of messages sent by node *prev*(k) last time this node had the right to transmit before the time window of duration Q_i^k ended. Reasoning in the same way, we obtain the more general expression:

$$\Omega^{y\to k}\left(Q_i^k\right)=\begin{cases}0, & if\ y=k\\ T_{MS}\times n_{slots}^{y\to k}\left(Q_i^k\right)+T_{PR}+\Omega^{next(y)\to k}\left(Q_i^k\right), & if\ y\neq k\end{cases} \tag{13}$$

where $n_{slots}^{y\to k}$ is a lower bound on the number of messages sent by node y at the last time node y held the right to transmit before the time window of duration Q_i^k ended. In order to find $n_{slots}^{y\to k}(Q_i^k)$ we compute a lower bound on the length of the output queue of node y at the last time node y held the right to transmit before the time window of duration Q_i^k ended. $LBql^{y\to k}$ denotes this lower bound. If we know $LBql^{y\to k}$, we can compute $n_{slots}^{y\to k}$.

$$n_{slots}^{y\to k}\left(Q_i^k\right)=\min\left\{mpc^y,\max\left\{\ 0,\ LBql^{y\to k}\left(Q_i^k\right)\right\}\ \right\} \tag{14}$$

A lower bound on the queue length must be 0 or more, hence the term $\max\{0, LBql^{y\to k}(Q_i^k)\}$ in (14). It represents another lower bound on the queue length. If, however, this would be greater than mpc^y, then $n_{slots}^{y\to k}=mpc^y$, because no more than mpc^y messages can be transmitted 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 $L^{y\to k}(Q_i^k)$ after the message from S_i^k was put in the output queue. $L^{y\to k}(Q_i^k)$ is given by:

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

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 $1\le ql^k$. Since the queue length of node k depends on the number of arrived messages and on the number of transmitted messages, we obtain:

$$ql^k \le \sum_{\forall S_j^k \, on \, N^k} \left\lfloor \frac{L^{y \to k}(Q_i^k)}{T_j^k} \right\rfloor - ntransmitted^k \tag{16}$$

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

$$ql^y \ge \sum_{\forall S_j^y \, on \, N^y} \left\lfloor \frac{L^{y \to k}(Q_i^k)}{T_j^y} \right\rfloor - ntransmitted^y \tag{17}$$

Observe that (16) and (17) 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 the difference between the number of TDMA cycles received by any two nodes is at most one. Hence:

$$nTDMAcycles^y \le nTDMAcycles^k + 1 \tag{18}$$

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 $L^{y \to k}(Q_i^k)$. This implies that all its TDMA cycles transmitted mpc^k messages. Therefore:

$$nTDMAcycles^k \le \left\lceil \frac{ntransmitted^k}{mpc^k} \right\rceil \tag{19}$$

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:

$$ntransmitted^y \le nTDMAcycles^y \times mpc^y \tag{20}$$

Combining (18), (19) and (20) yields:

$$ntransmitted^y \le \left(\left\lceil \frac{ntransmitted^k}{mpc^k} \right\rceil + 1 \right) \times mpc^y \tag{21}$$

We have already seen that $1 \leq ql^k$. Combining it with (16), (17) and (21) leads to (22).

$$LBql^{y\to k}(Q_i^k) = \sum_{\forall S_j^y on N^y} \left\lfloor \frac{L^{y\to k}\left(Q_i^k\right)}{T_j^y} \right\rfloor - \left(\left\lceil \frac{\left(\sum_{\forall S_j^k on N^k} \left\lceil \frac{L^{y\to k}\left(Q_i^k\right)}{T_j^k} \right\rceil \right) - 1}{mpc^k} \right\rceil + 1 \right) \times mpc^y \tag{22}$$

The expression for $LBql^{y\to k}$ in (22) can be used in (14) and then in (13) to obtain the value of $\Omega^{y\to k}$. Note that $\Omega^{y\to k}$ can be computed without circular dependencies because $\Omega^{k\to k}$ is computed from (13) and then $\Omega^{prev(k)\to k}$ is computed based on $\Omega^{k\to k}$. $\Omega^{prev(prev(k))\to k}$ is computed based $\Omega^{prev(k)\to k}$, and so on.

4. Exact Analysis

We will now develop an exact analysis of TDMA/SS. The approach is to use results from Section 2 to find the worst-case queuing time of a message stream and simulate scheduling in order to decide if the system meets all deadlines. We will describe the algorithm to determine the length Lbp^k of the busy period, and the queuing time for a given stream.

The length Lbp^k of the busy period is found by developing the timing behaviour, departing from the time instant that maximizes the amount of interference caused by higher-priority message streams.

The queuing time is found similarly by developing the timing behaviour of the network, departing from an initial state such that the maximum queuing time from of the given stream is found. For this initial state, it employs the critical instant definition as described in Section 2. Thus, the queuing time resulting from the time evolution of the protocol departing from this instant, found within the length Lbp^k of the busy period will be the maximum queuing time.

4.1. Overview

Algorithm 1 shows how to develop the timing behaviour of the network. This straightforward algorithm introduces the main steps necessary to do this. We start by setting up the initial state of the

Algorithm 1. Develop the timing behaviour of the network departing from a defined initial state

```
1.  begin
2.    Setup initial state of the network;
3.    time ← 0;
4.    address_counter ← 1;
5.    loop
6.      Put messages from streams activated until current time in the respective output queue;
7.      Try to take up to mpc^address_counter messages from node^address_counter output queue;
8.      Increase time according to message queue state and medium access rules;
9.      address_counter ← next(address_counter );
10.   until time ≥ MAX_TIME;
11. end
```

Algorithm 2. Find the length Lbp^k of the busy period

```
1.  input
2.    k – the index of the node;
3.  begin
4.    Setup initial state of the network;
5.    time ← 0;
6.    address_counter ← next( k );
7.    Lbp^k ← 0;
8.    loop
9.      Put messages from streams activated until current time in the respective output queue;
10.     if address_counter = k and the output queue of node k is empty then
11.       Lbp^k ← time;
12.     else
13.       Try to take up to mpc^address_counter messages from node^address_counter output queue;
14.       Increase time according to message queue state and medium access rules;
15.       address_counter ← next( k );
16.     end if
17.   until Lbp^k > 0 or time ≥ lcm( ∀T_i^k on node^k ) ;
18.   if Lbp^k > 0 then return Lbp^k; else return lcm ( ∀T_i^k on node^k ); end if
19. end
```

network. Then we establish, for now, that the node starting to access the network is node 1, and enter a loop where the simulated time is developed according to the medium access rules. At each step of the loop, we will check for messages that where activated until the current time, delivering messages of these message streams to the respective node's output queue and maintain the state of the different output queues. By checking the current state of the queues, we decide how to make the time evolve.

In this case, Algorithm 1 will develop the timing behaviour of the network for a pre-defined amount of time (defined by `MAX_TIME`). Algorithm 1 conveys the main idea we will employ. In the following sections we present the same basic structure of the algorithm, with the necessary changes to find the length of the busy period and the queuing time for a given stream.

4.2. Algorithm to Find the Length of the Busy Period

To find the length Lbp^k of the busy period we will follow the general structure of Algorithm 1. In this case, we have a different stopping condition. Recalling the definition of busy period: for the most demanding arrival pattern,

Algorithm 3. Compute the queuing time of S_i^k

```
1.  input
2.    k  – the node index where the stream for which we will compute the queuing time;
3.    i  – the index of the stream for which we will compute the queuing time;
4.  begin
5.    max_queuing_time ← FAILURE;
6.    Compute the set of activation times A_i^k;
7.    for all activation_time ∈ A_i^k
8.      Setup initial state of the network;
9.      Set activation time of S_i^k to activation_time
10.     time ← 0;
11.     address_counter ← k;
12.     loop
13.       if (S_i^k is activated in this cycle) then
14.           Compute blocking time B_i^k;
15.           time ← time + B_i^k;
16.           address_counter ← next( k );
17.       end if
18.       Put messages from streams activated until current time in the respective output queue;
19.       Try to take up to mpc^address_counter messages from node^address_counter output queue;
20.       Increase time according to message queue state and medium access rules;
21.       address_counter ← next( address_counter );
22.     until (current_time – activation_time) > deadline of S_i^k or a message from S_i^k is removed from node k's output queue
23.     if a message from S_i^k was removed from N^k's output queue and (current_time–activation_time)>max_queuing_time then
24.       max_queuing_time ← current_time – activation_time;
25.     end if
26.   end for
27.   return max_queuing_time;
28. end
```

the length of the busy period will be from $t = 0$ up to the first idle time. So we merely have to develop the timing behaviour of the network, until we find a turn of node k where its message queue is empty, as shown by Algorithm 2.

To setup the initial state of the network, we follow Condition 2) from Section 2 and also consider the case where all message streams on node k arrive simultaneously, which maximizes the amount of interference caused by higher-priority streams. Looking at line 17 from Algorithm 2, we can see that the loop will run until a value for Lbp^k is found or, to protect from cases where there is no idle time, the loop stops when the simulated time is more than the least common multiple (*lcm*) of the periods from all streams in node k.

4.3. Algorithm to Find the Maximum Queuing Time

To determine the maximum queuing time of a stream we can adopt a similar approach. To do this, we simply determine the set of activation times A^k to test and add a loop that will execute for each of these activation times (Algorithm 3, lines 7 to 26). In line 23 of Algorithm 3 we check if the obtained queuing time is the maximum so far, so that, when we have tried all the activation times in set A^k, we will have the maximum queuing time.

Algorithm 4. Increase time according to message queue state and medium access rules

```
1.  input
2.   address_counter      – the node index of the node currently holding the right to access the medium
3.   k                    – the node index where the stream which we will compute the queuing time
4.   i                    – the index of the stream which we will compute the queuing time
5.  begin
6.      for all messages M from node's N^address_counter_r message queue up to mpc^address_counter
7.          Remove highest priority message M from current node's output queue;
8.          if M is not a message from S_i^k then
9.              time ← time + T_MS;
10.         end if
11.     end for
12.     time ← time + T_PR;
13. end
```

In Algorithm 3, the stopping condition for the development of the timing behaviour (line 22) was also modified. Now this loop is run until a message from the stream for which we will compute the queuing time is sent, or until its deadline is exceeded. Another difference introduced was that the insertion of blocking time. The blocking time must be introduced in the time instant preceding the activation of the stream for which we will compute the queuing time. The blocking time is computed by Equation (11). The blocking time is inserted by increasing the simulated time and changing `address_counter` to the next node.

4.4. Detailing the Algorithms

This section will present further details for the most important components of the algorithms presented previously.

Compute the set of activation times A_i^k – The first component we will detail here is the step to compute the set of activation times A_i^k. This set will depend on the scheduling employed to the output queues. The activation times set will be defined according to Equation (6).

Setup initial state of the network – To setup the initial state of the network, we employ the critical instant as defined previously. Remember that our definition of Φ in Equation (5) returns the amount of time that messages must be synchronously released in each node.

Note that, in Algorithm 3, the activation time of the stream for which we will compute the queuing time is set again for each activation time.

$$network = \left(5, \left\{N^1, N^2, N^3, N^4, N^5\right\}, 1, 1/5\right) \tag{23}$$

$$\begin{cases} N^1 = \left(4, \left\{S_1^1, S_2^1, S_3^1, S_4^1\right\}, 2\right) \\ \begin{cases} T_1^1 = D_1^1 = 8 \\ T_2^1 = D_2^1 = 16 \\ T_3^1 = D_3^1 = 25 \\ T_4^1 = D_4^1 = 100 \end{cases} \end{cases} \begin{cases} N^2 = \left(3, \left\{S_1^2, S_2^2, S_3^2\right\}, 1\right) \\ \begin{cases} T_1^2 = D_1^2 = 12 \\ T_2^2 = D_2^2 = 35 \\ T_3^2 = D_3^2 = 140 \end{cases} \end{cases} \begin{cases} N^3 = \left(2, \left\{S_1^3, S_2^3\right\}, 1\right) \\ \begin{cases} T_1^3 = T_1^3 = 9 \\ T_2^3 = T_2^3 = 50 \end{cases} \end{cases}$$

$$\begin{cases} N^4 = \left(5, \left\{S_1^4, S_2^4, S_3^4, S_4^4, S_5^4\right\}, 2\right) \\ \begin{cases} T_1^4 = D_1^4 = 15 \\ T_2^4 = D_2^4 = 20 \\ T_3^4 = D_3^4 = 30 \\ T_4^4 = D_4^4 = 100 \\ T_5^4 = D_5^4 = 150 \end{cases} \end{cases} \begin{cases} N^5 = \left(2, \left\{S_1^5, S_2^5\right\}, 1\right) \\ \begin{cases} T_1^5 = D_1^5 = 33 \\ T_2^5 = D_2^5 = 56 \end{cases} \end{cases} \tag{24}$$

Figure 5. Example of a network scenario (1).

Put messages from streams activated until current time in the respective output queue – This is done by checking all streams in the network for which the activation time has elapsed. Streams in this condition will generate a message to be put in the respective node's output queue. Additionally, the activation time of the stream is set to its next period.

Increase time according to message queue state and medium access rules – To develop time, the state of the output queue from the node currently holding the right to access the medium is verified. For each message in the output queue, up to $mpc^{address\_counter}$, the time is increased by T_{MS}. At the end of the node's turn, the time is increased by T_{PR}. Algorithm 4 illustrates this procedure.

4.5. Numerical Example

Let us put forward a demonstration scenario that will enable us to better grasp the algorithm behaviour. Equations (23) and (24) in Figure 5 describe this demonstration scenario.

Figure 6 presents the network schedule for $node^2$. It is also possible to observe the evolution of the node's queue and the queuing time for S_2^2.

This network schedule depicts exactly the algorithm's behaviour. When the stream given as input for the algorithm is S_2^2, the algorithm will develop time by simulating the network schedule and produce the exact time evolution as depicted in Figure 6, when the activation time to be tested for S_2^2 is 0, which is the activation time leading to the maximum queuing delay.

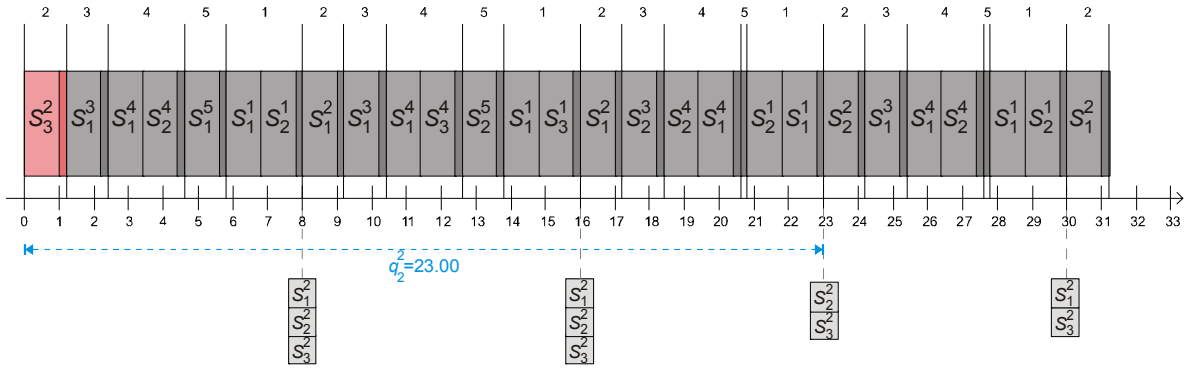

Figure 6. Network schedule for demonstration scenario (1).

Table 1. Queuing times for demonstration scenario (1).

Node	*Stream*	*Q* *(time units)*	*q* *(time units)*
N^1	S_1^1	8	8
	S_2^1	16	9
	S_3^1	16	16
	S_4^1	45	40
N^2	S_1^2	8	8
	S_2^2	24	23
	S_3^2	57	35
N^3	S_1^3	8	8
	S_2^3	45	32

Node	*Stream*	*Q* *(time units)*	*q* *(time units)*
N^4	S_1^4	8	8
	S_2^4	16	9
	S_3^4	17	16
	S_4^4	24	16
	S_5^4	29	27
N^5	S_1^5	8	8
	S_2^5	15	15

The algorithm will perform from time 0 to time 30. At time 30, the algorithm will verify that the S_2^2 given as input was scheduled to be sent, and therefore it will exit returning the current time value.

Using an implementation of the algorithm, we can obtain the resulting queuing times. Table 1 presents the queuing times for all messages in this scenario resulting from the algorithm presented (column labelled q). The column labelled Q contains the calculated queuing times using the analysis in Section 3. We can see that our algorithm for the exact analysis presents tighter results than the non-exact analysis. This is expected.

5. Discussion and Related Work

TDMA/SS has the following advantages. First, TDMA/SS does not require sensing-while-transmitting. Second, 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`. This is a relevant practical aspect that facilitates the acceptance of TDMA/SS for small, inexpensive embedded devices with bare hardware resources. Implementations of TDMA/SS only require this timer, as opposed to other TDMA protocols that typically need more timers. This was also one of the advantages of the design of BuST [13], a token protocol with hard real-time characteristics and budget sharing, with the features related to TDMA/SS, Third, TDMA/SS is resilient to crashes if nodes are fail-silent. (One way to implement TDMA/SS, as was done in P-NET standard [4], 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) assuming FIFO scheduling on each node, (ii) lacking an accurate calculation of Ω, and (iii) lacking the opportunity to transmit multiple messages per TDMA cycle.

The TDMA/SS protocol is similar to the ARINC 629 protocol in that ARINC 629 is a TDMA protocol which does not need synchronized clocks. Nodes are given time slots in a pre-specified order; they have a terminal gap (TG) specifying an idle time interval between nodes (similar to our T_{PR}) and they permit slot skipping. Unfortunately, the only available analysis of ARINC 629 [12] is not accurate in the sense that it does 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 [14] 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, synchronized clocks are required.

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 [15-17]. 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 [10, 11] addresses a problem similar to ours (the S_p in [10] is equivalent to our mpc^p; in [11] mpc^k is more restricted, it is assumed to be 1). However, neither [10] nor [11] 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 was studied in [18]. It uses explicitly message passing where a token must circulate and nodes announce their priority before transmitting. That is unlike TDMA/SS which only prioritises messages on each node.

Real-time scheduling on networks with explicit token bus was also studied in [19]. Each node is given a budget and if the hard real-time traffic (called synchronous traffic) on a node requests less than the budget then the remaining capacity is made available for non-real-time traffic. The analysis of TDMA/SS that we present in this paper is different however in that in TDMA/SS, an unused slot makes capacity available for hard real-time traffic at another slot and the amount of that capacity made available is calculated with our analysis ot TDMA/SS.

Implicit EDF is a TDMA MAC protocol proposed for wireless channels [20, 21]. Although its operation is very different from TDMA/SS, Implicit EDF shares many of the advantages offered by TDMA/SS in that (i) both of them are collision-free yet they do not need to store the entire TDMA schedule, (ii) they do not depend on synchronized clocks and (iii) they can operate even in the presence of certain node failures (crash failures). One key difference however is that the operation of the TDMA/SS MAC protocol does not require nodes to know all messages streams in the system and consequently it is easy to add new message streams and/or nodes to the system. In fact, this was one of the main motivation for the P-NET standard [4]; a standard that uses TDMA/SS with SMTC.

An important feature of our analysis of TDMA/SS in this paper is that a node k can have $mpc^k>1$. This feature ($mpc^k>1$) is useful because it reduces the amount of time that the network spends on sending a protocol slot. But naturally, this feature begs the question: How should *mpc*:s be assigned to nodes such that all deadlines are met? A simple approach is as follows. Initially assign $mpc^k=1$ to each node k. Perform a schedulability test and record which nodes had a message stream that missed a deadlines. For those nodes, increase *mpc* by one. As long as there is a deadline miss and T_{TDMA} does not exceed the minimum D_i of all message streams then continue to perform schedulability tests and increase mpc^k for those nodes that missed a deadline. An interesting aspects of this simple algorithm to assign mpc^k is that it can be performed by replacing the schedulability test by run-time monitoring of deadline misses and this makes the algorithm for assigning *mpc*:s to nodes fully decentralized.

6. Conclusions and Future Work

TDMA/SS represents an important class of TDMA networks, implemented in COTS hardware, and it is suited for real-time applications. We have presented an analysis of TDMA/SS for DM scheduled output queues and also an algorithm that computes exact queuing times for TDMA/SS. The algorithm was based on simulation. We left open the questions (i) whether it is possible to formulate exact schedulability

conditions as a set of inequalities and (ii) how to perform approximate schedulability [22] analysis for TDMA/SS and in that way achieve a polynomial time-complexity.

References

[1] H. Kopetz and G. Grunsteidl, "TTP-a protocol for fault-tolerant real-time systems," IEEE Computer, vol. 27, 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 12th Euromicro Conference on Real-Time Systems (ECRTS'00), 2000, pp. 239-246.

[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, 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, pp. 229-249, May 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, pp. 46-61, 1973.

[7] L. George, N. Rivierre, and M. Spuri, "Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling," INRIA, Technical Report RR-2966, September 1996.

[8] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller Area Network (CAN) schedulability analysis: Refuted, Revisited and Revised," Real-Time Systems, vol. 35, pp. 239-272, 2007.

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

[10] K. Tindell, "Analysis of Hard Real-Time Communications," Real-Time Systems Journal, vol. 9, pp. 147 - 171, 1995.

[11] M. Spuri, "Holistic Analysis for Deadline Scheduled Real-Time Distributed Systems," INRIA, Technical Report RR-2873, April 1996.

[12] 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.

[13] G. Franchino, G. C. Buttazzo, and T. Facchinetti, "BuST: Budget Sharing Token protocol for hard real-time communication," in Emerging Technologies & Factory Automation, 2007. ETFA. IEEE Conference on, 2007, pp. 1278-1285.

[14] 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 24th IEEE International Real-Time Systems Symposium (RTSS'03), 2003, p. 298.

[15] 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, pp. 729 - 741, December 1995 1995.

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

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

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

[19] G. Franchino, G. Buttazzo, and T. Facchinetti, "BuST: Budget Sharing Token Protocol for Hard Real-Time Communication," in 12th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2007), Patras, Greece, 2007.

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

[21] T. L. Crenshaw, A. Tirumala, S. Hoke, and M. Caccamo, "A Robust Implicit Access Protocol for Real-Time Wireless Collaboration," in 17th Euromicro Conference on Real-Time Systems (ECRTS'05), Palma de Mallorca, Balearic Islands, Spain, 2005, pp. 177 - 186.

[22] S. Chakraborty, S. Künzli, and L. Thiele, "Approximate Schedulability Analysis," in 23rd IEEE International Real-Time Systems Symposium (RTSS'02), Austin, Texas, 2002.

Appendix A – List of Symbols

Symbol	Meaning
T_{MS}	Time slot corresponding the transmission of one message
T_{PR}	Time slot corresponding to a message signaling that the node will not transmit any more messages during the current TDMA cycle
net	A TDMA/SS network
N^k	A node k, of the TDMA/SS network
S_i^k	A message stream i at node k
ns^k	Number of message streams in node k
$sched\_policy^k$	Policy adopted for scheduling messages at node k's local queue
msg_queue^k	Message queue of node k
mpc^k	Messages per (TDMA) cycle allowed for node k
T_i^k	Periodicity at which a message related to a stream S_i^k is queued
D_i^k	Relative deadline of S_i^k.
q_i^k	Actual or obtained with the exact analysis maximum queuing time of messages belonging to S_i^k
r_i^k	Maximum response time of messages belonging to S_i^k
Q_i^k	Upper bound on the queuing time of messages belonging to S_i^k
R_i^k	Upper bound on the response time of messages belonging to S_i^k
$prev(k)$	Previous node (obtained by circularly incrementing the current address)
$next(k)$	Next node (obtained by circularly incrementing the current address)
$hp^k(S_i^k)$	Subset of messages streams on node k with higher priority than S_i^k
$lp^k(S_i^k)$	Subset of messages streams on node k with lower priority than S_i^k
`address_counter`	Variable that keeps track of the node holding the right to transmit
$\Phi^{y \to k}$	Amount of time which messages from nodes y should arrive earlier in order to delay the response times of the messages in node k the most
Lbp^k	Length of the busy period of a node k
T_{TDMA}	TDMA cycle duration when no slots are skipped
B_i^k	Blocking factor due to the non-preemptable nature of message transmissions and TDMA cycle
$nss^{y \to k}(t, i)$	Lower bound on the number of skipped slots on node y during a time interval of length t
$slot_i^k(Q_i^k)$	Number of time slots needed during Q_i^k by message streams of higher priority than S_i^k
$\Omega^{y \to k}(t)$	Lower bound on the amount that the window of node y should be shrunk at the end of time t (in order to avoid considering more messages than those that actual cause interference)
$LBql^{y \to k}(t)$	Lower bound on the queue length at node y during a time interval of duration t
$n_{slots}^{y \to k}(t)$	Lower bound on the number of messages transmitted during a time interval of duration t
$L^{y \to k}(t)$	Amount of time after the message from S_i^k was put in the output queue
ql^y	Length of the output queue of node y at time $L^{y \to k}$
$ntransmitted^k$	Number of messages transmitted during the time window of length $L^{y \to k}$