Exact Analysis of TDMA with Slot Skipping

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

Abstract

Consider a communication medium shared among a set of computer nodes; these computer nodes issue messages that are requested to be transmitted and they must finish their transmission before their respective deadlines. TDMA/SS is a protocol that solves this problem; it is a specific type of Time Division Multiple Access (TDMA) where a computer node is allowed to skip its time slot and then this time slot can be used by another computer node. We present an algorithm that computes exact queuing times for TDMA/SS in conjunction with Rate-Monotonic (RM) or Earliest-Deadline-First (EDF).

1. Introduction

Achieving end-to-end real-time guarantees requires that the communication network supports bounded-delay delivery of inter-task messages. To achieve this, one commonly-used strategy for controlling the access of a shared medium is to assign a time slot to each node in the network and nodes take turns to access the medium. This medium access control, named Time Division Multiple Access (TDMA) is widely exploited and a wealth of research results is available [1-3].

The static nature of TDMA offers several advantages such as (i) low run-time overhead, (ii) good composability in terms of timeliness; (iii) low fault-detection latency and (iv) the schedulability analysis for real-time traffic becomes trivial. Unfortunately, the very same static nature brings reduced flexibility implying that (i) unused slots are wasted, (ii) it is difficult to add new computer nodes to the system without changing the static schedule; and (iii) for periodic real-time traffic the amount of memory required for storing the static schedule may be very large for certain periods, such as prime numbers. In order to remove and mitigate those drawbacks, we consider TDMA protocols with slot skipping (TDMA/SS) as presented in [4]. In this particular class of TDMA networks, a slot is skipped when it is not used, hence the next slot can start earlier and, by this, reclaim time for hard real-time traffic. Such an approach is already in use in some commercial-off-the-shelf (COTS) technology [5] and a schedulability analysis that takes slot skipping into account is also presented in [4]. Unfortunately, that analysis [4] was not exact.

In this paper we present an exact analysis of TDMA/SS assuming that: (i) the message streams on each node are known; and (ii) each node schedules messages in their output queue according to one of the real-time scheduling policies Rate-Monotonic (RM) or Earliest-Deadline-First (EDF). The approach is to use results from previous works [4, 6] to find a critical instant of a message stream (when analyzing RM) or a message (when analyzing EDF), and simulate scheduling in order to decide if the system meets all deadlines. For EDF, this technique is very time-consuming since critical instants of every message must be explored. We propose a technique to reduce the time-complexity. Our new algorithm that analyses RM has pseudo-polynomial time complexity and our new algorithm that analyses EDF has exponential time-complexity. Nonetheless, these algorithms are the first algorithms that offer exact analysis of TDMA/SS.

The rest of the paper is organized as follows. Section 2 presents the system model and the operation of TDMA/SS networks. It also presents conditions for critical instants and other key observations. These observations are then used, in Section 3, which presents the new algorithm for both RM and EDF. Two numerical examples are also given to illustrate the new algorithms. Finally, Section 4 gives conclusions.

2. TDMA Networks with Slot Skipping (TDMA/SS)

2.1. System Model

The network is composed of a set of n nodes, communicating messages over a shared medium. Contention between nodes is resolved by a Time Division Multiple Access (TDMA) control scheme. The access to the medium is ordered by time, such that each node is assigned one or more time slots (message slots), each of length T_{MS}, in a cyclic schedule – the TDMA cycle. When a node observes its turn to send, it may transmit up to the number of message slots assigned to it. At the time a node either uses all the message slots assigned or has no more messages in its

outgoing queue, it indicates that it has finished transmitting in the current TDMA cycle by using a protocol slot of length $T_{PR} << T_{MS}$.

The network model is defined as follows:

$$Network = (n, \{node^1, node^2, ..., node^n\}, T_{MS}, T_{PR}) \quad \textbf{(1)}$$

Each node k (k = 1, ..., n) possesses a set $\{S_1^k, S_2^k, \ldots, S_{ns}^k\}$ of ns message streams. Each node has associated an $mpc^k > 0$ (messages per cycle) parameter, defining the maximum number of messages the node is permitted to transmit in a TDMA cycle. The algorithm will also maintain a queue msg_queue^k holding pending (outgoing) messages that belong to the set of message streams associated with each node k. A node in our network model is defined as follows:

$$node^k = (ns^k, \{S_1^k, S_2^k, ..., S_{ns^k}^k\}, msg\_queue^k, sched\_policy^k, mpc^k) \quad \textbf{(2)}$$

where $sched\_policy^k$ is the policy adopted for scheduling messages from msg_queue^k. This can be one of two values: RM for Rate Monotonic or EDF for Earliest Deadline First.

Each message stream is characterized by the periodicity at which a message related to the respective stream is queued to be transmitted on the network, and the corresponding relative deadline (T_i^k and D_i^k, respectively). It is assumed that $D_i^k \leq T_i^k$ and all messages in the network have length T_{MS}. The algorithm will maintain $act\_time_i^k$, holding the last time instant at which a message related to S_i^k was put into the msg_queue^k. Hence, a stream is defined as follows:

$$S_i^k = (T_i^k, D_i^k, act\_time_i^k) \quad \textbf{(3)}$$

Every message may need to be queued before it is transmitted. 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 deadline. We are interested in finding a bound on the queuing time of streams, and consequently determining whether all messages meet their deadlines.

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

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

And in the analysis it is useful to know the T_{TDMA}, defined as follows:

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

$$network = (3, \{node^1, node^2, node^3\}, 1, 1/5) \quad \textbf{(4)}$$

$$\begin{cases} node^1 = (3, \{S_1^1, S_2^1, S_3^1\}, \phi, RM, 1) \\ \quad \begin{cases} S_1^1 = (4,4,0) \\ S_2^1 = (13,13,0) \\ S_3^1 = (13.4,13.4,0) \end{cases} \\ node^2 = (1, \{S_1^2\}, \phi, RM, 1) \quad \begin{cases} S_1^2 = (5.2,5.2,0) \end{cases} \\ node^3 = (1, \{S_1^3\}, \phi, RM, 1) \quad \begin{cases} S_1^3 = (7,7,0) \end{cases} \end{cases} \quad \textbf{(5)}$$

Figure 1. Example of a network scenario.

If we are analysing a message stream S_i^k and it turns out that $ns^k \leq mpc^k$ then the analysis becomes trivial. For this reason, when we analyse S_i^k we will assume that $ns^k \geq mpc^k + 1$.

2.2. Network Example and Illustration of Operation

We will now describe the operation of the network protocol being used. All nodes maintain a variable – `address_counter` – that keeps track of the current node that has the right to transmit. The `address_counter` has the same value on all nodes, so in the discussion we treat it as one variable. When `address_counter` makes the transition to k, then node k dequeues mpc^k messages from its output queue, transmits those mpc^k messages, transmits a protocol slot, and `address_counter` becomes the address of the next node in the network. If the output queue contains $0 \leq x < mpc^k$ messages then only those $0 \leq x$ messages are transmitted (we say that node k skips $mpc^k - x$ slots), then a protocol slot is transmitted (this takes T_{PR} time units), and then the `address_counter` is modified as before.

When a node does not transmit, it listens to the network to update the `address_counter` to be consistent with the other nodes. In order for this to be possible, we assume that all nodes hear the same state of the network.

As an instantiation of (1), (2) and (3) concerning the previously described network and message models, consider the network with 3 nodes given by Equations 4 and 5 in Figure 1.

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 that initiated transmission at time instant 10.4.

2.3. Worst-Case Response Times

Recall that our goal is to decide if a given set of message streams meets deadlines on a given network scheduled by TDMA/SS when each computer node schedules messages locally using RM or EDF. But before dwelling into that analysis, let us discuss analysis of scheduling tasks using RM and EDF on a single processor. For RM, such analysis is commonly performed using the notion of a *critical instant*. A critical instant of a task in RM is an instant such that the response time of the task is maximized if it arrives at that instant. Hence, the response time of a task scheduled by RM on a single processor is found simply by analyzing the response time of the task when it arrives at a critical instant (note that later results for non-preemptive scheduling as found in [7] and [8] do not apply in the case of TDMS/SS networks [6]).

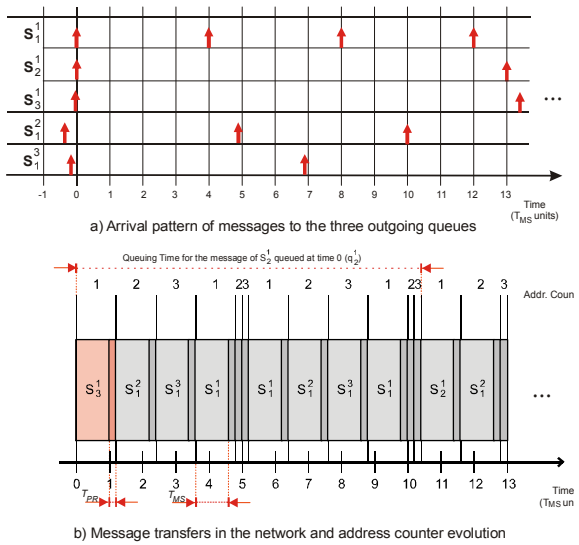

Figure 2. Example message arrival patterns and corresponding network schedules.

For EDF, the notion of critical instant is usually not used. However, observe that EDF is a job-static priority scheduling algorithm, meaning that although the relative priority order among a set of tasks may change with time, the relative priority order of jobs do not change. For this reason, every job scheduled by EDF can be analyzed in a similar way as a task is analyzed by RM.

2.3.1. Worst-Case Response Times with RM

To characterize the critical instant under RM scheduled output queues in TDMA/SS, it is tempting to reuse the condition for the critical instant as used in RM 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 one message on the same node as S_i^k arrives marginally before and this message is transmitted and causes blocking. However, consider the example shown in Figure 2 and analyze the queuing delay of the message from S_2^1 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_2^1 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 RM on a single processor. It was in fact shown [6] that the condition for the 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. infinitesimally before time t, a message stream (which is not S_i^k) released a message and started to transmit this message;
3. all other message streams in node k are activated synchronously, at time t;
4. all other message streams in each node y ($\forall y : 1 \leq y \leq ns^k \; y \neq k$) are synchronously activated $\Phi^{y \to k}$ time units before t.

It is easy to show that this $\Phi^{y \to k}$ time quantity before the start of the busy period in node k corresponds to consider that node's y busy period started in the previous TDMA cycle. As node addresses correspond to the cyclic sequence of assigned slots, this quantity may be given as follows [4]:

$$\Phi^{y \to k} = \begin{cases} 0 & if \; y = k \\ T_{PR} + \Phi^{next(y) \to k} & if \; y \neq k \end{cases} \quad \textbf{(6)}$$

Not only are these the conditions that maximize the queuing time from message requests under RM scheduled output queues, but also, conditions 3) and 4) maximize the amount of interference caused by higher priority streams for any scheduling policy applied to the output queues.

2.3.2. Worst-Case Response Times with EDF

In the case of EDF scheduled output queues, the same general reasoning applies. However, Condition 3) does not necessarily apply because, locally to each node, the critical instant does not always happen when all messages are activated synchronously. Assuming that we are interested in finding the critical instant for a S_i^k, the difference is that all messages in node k are activated synchronously, with the exception of S_i^k. The maximum queuing time of messages from S_i^k is found when the message stream is activated at a time a, within the length L^k of the busy period. (Observe that we compute L^k for every node.)

Given this, under EDF, Condition 2) is also changed, because one must account for a possible blocking after the activation of the message stream at time a. Thus, we must consider that, at time a, node k just started to transmit a last pending message. However, Equation 6 still holds. Under EDF, Condition 4) is all the same valid.

To find the queuing time for S_i^k would be necessary to analyze all schedule patterns resulting from the different activation times $a \in [0, L^k)$ of S_i^k, and verify which one that results in the highest queuing time. Clearly, this would be computationally too expensive. We know however, that not all of these schedules need to be explored in order to find the maximum queuing time (see the idea in [9]). Specifically, only the set of a values, for which $a + D_i$ coincides with at least the absolute deadline of a message from another stream. Then, for a message stream S_i^k, the set of values A_i^k is given as follows:

$$A_i^k = \bigcup_{j=1}^{ns^k} \left\{ c \times T_j + D_j - D_i : c \geq 0 \right\} \cap \left[0, L^k \right) \quad \textbf{(7)}$$

We can now find the maximum queuing time for a S_i^k by analysing the schedule patterns resulting from the different activation times $a \in \{A\}$, given by:

$$Q_i^k = \max \left\{ q_i^k(a) - a \right\} \quad \textbf{(8)}$$

where $q_i^k(a)$ is the time from the beginning of the busy period until S_i^k arrives.

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

Another alternative to find L^k is trying to iteratively compute the size of the busy period. The iteration starts with the minimum amount of time necessary to transmit one instance of each message stream in node k. The following iterations will account for the time spent to transmit messages that arrived before time t, given by $W^k(t)$:

$$\begin{cases} L^{k(0)} = \left\lfloor \dfrac{ns^k}{mpc^k} \right\rfloor \times T_{TDMA} \\ L^{k(m+1)} = W^k(L^{K(m)}) \end{cases} \quad \textbf{(9)}$$

To compute $W^k(t)$, we may reason that $\lceil t / T_i^k \rceil$ is the number of messages from each stream i in node k that arrived to the output queue until the given time t. We can then reason that to transmit x messages, a node k takes $\lfloor x / mpc^k \rfloor$ TDMA cycles, and it also needs to wait for $x \; mod \; mpc^k$ message slots. Therefore:

$$W^k(t) = \left\lfloor \frac{\sum_{\forall S_i^k \; on \; N^k} \left\lceil \dfrac{t}{T_i^k} \right\rceil}{mpc^k} \right\rfloor \times T_{TDMA} + \left(\left(\sum_{\forall S_i^k \; on \; N^k} \left\lceil \frac{t}{T_i^k} \right\rceil \right) \bmod mpc^k \right) \times T_{MS} \quad \textbf{(10)}$$

While computing L^k by Equation 10 and 11 will give in much more accurate results, Equation 11 does not account for skipped slots. Thus, for a set of streams that are in fact schedulable by the system, Equation 10 may never converge.

It would be feasible to exploit the analysis of TDMA/SS networks in [10] to compute the length of the busy period. Nonetheless, to find the length L^k of the busy period, we will also exploit the idea of using an algorithm that imitates TDMA/SS networks over time. The benefits of doing so are twofold. First, 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( k );
10   until time ≥ MAX_TIME;
11  end
```

Algorithm 2. Find the length L^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     L^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        L^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 L^k > 0 or time ≥ lcm (∀T_i^k on node^k ) ;
18    if L^k > 0 then
19      return L^k;
20    else
21      return lcm (∀T_i^k on node^k ) ;
22   end if
23  end
```

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 each 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( k );
22      until current time > deadline of S_i^k or a message from S_i^k is removed from node k's outgoing queue
23      if (current_time - activation_time > max_queuing_time or max_queuing_time = FAILURE) then
24        max_queuing_time ← current_time - activation_time;
25      end if
26    end for
27    return max_queuing_time;
28   end
```

algorithm to determine the queuing times is very similar to the one for finding the length of the busy period. Second, by using the algorithm we avoid some of the assumptions introduced by the analysis in [10], thus this will yield more tight results.

3. Exact algorithm

This section will describe the algorithm to determine the length L^k of the busy period, and the queuing time for a given stream for both RM and EDF scheduled output queues.

The length L^k of the busy period is found by developing the timing behavior of the network, departing from the time instant that maximizes the amount of interference caused by higher priority streams (see Section 2.3.1).

The queuing time is found similarly by developing the timing behavior 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 previous sections. Thus, the queuing time resulting from the time evolution of the protocol departing from this instant, found within the length L^k of the busy period will be the maximum queuing time.

3.1. Overview

An algorithm to develop the timing behavior of the network is sketched in Algorithm 1. This straightforward algorithm introduces the main steps necessary to do this. We start by setting up the initial state of the 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 instances 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 behavior 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.

3.2. Algorithm to Find the Length of the Busy Period

To find the length L^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, 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 behavior 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 Conditions 2) and 3), which maximize 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 L^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; Algorithm 2, line 17.

3.3. Algorithm to Find the Maximum Queuing Time

To determine the maximum queuing time of a stream we can adopt a similar approach. However, for the case of EDF scheduled output queues, we must develop the timing behavior of the network for each of the activation times that might result in the worst queuing time. 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 25). In line 22 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 3 is valid for both RM and EDF scheduled output queues. The necessary distinctions that need to be made are included in this algorithm and details in the next section.

In Algorithm 3, the stopping condition for the development of the timing behavior (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 6, where the function `lp^k()` is according to the scheduling method applied in the output queue of node k. The blocking time is inserted by increasing the simulated time and changing `address_counter` to the next node.

3.4. 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. If the scheduling is RM, the activation times set will be {0}. In the case of EDF scheduling the set of activation time will be defined according to Equation 9.

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 6 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.

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

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

```
1  input
2    address_counter – the node index of the node
     currentIy 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 each message from node^address_counter message queue
     up to mpc^address_counter
7      Remove highest priority message M from
       current_node's outgoing 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
```

$$network = \left(5, \left\{node^1, node^2, node^3, node^4, node^5\right\}, 1, 1/5\right) \quad (12)$$

$$\begin{cases} node^1 = \left(4, \left\{S_1^1, S_2^1, S_3^1, S_3^1\right\}, \phi, RM, 2\right) \\ \begin{cases} S_1^1 = (8,8,0) \\ S_2^1 = (16,16,0) \\ S_3^1 = (25,25,0) \\ S_4^1 = (100,100,0) \end{cases} \end{cases} \begin{cases} node^2 = \left(3, \left\{S_1^2, S_2^2, S_2^2\right\}, \phi, RM, 1\right) \\ \begin{cases} S_1^2 = (12,12,0) \\ S_2^2 = (35,35,0) \\ S_3^2 = (140,140,0) \end{cases} \end{cases} \begin{cases} node^3 = \left(2, \left\{S_1^3, S_2^3\right\}, \phi, RM, 1\right) \\ \begin{cases} S_1^3 = (9,9,0) \\ S_2^3 = (50,50,0) \end{cases} \end{cases}$$

$$\begin{cases} node^4 = \left(5, \left\{S_1^4, S_1^4, S_2^4, S_3^4, S_3^4\right\}, \phi, RM, 2\right) \\ \begin{cases} S_1^4 = (15,15,0) \\ S_2^4 = (20,20,0) \\ S_3^4 = (30,30,0) \\ S_4^4 = (100,100,0) \\ S_5^4 = (150,150,0) \end{cases} \end{cases} \begin{cases} node^5 = \left(2, \left\{S_1^5, S_2^5\right\}, \phi, RM, 1\right) \\ \begin{cases} S_1^5 = (33,33,0) \\ S_2^5 = (56,56,0) \end{cases} \end{cases} \quad (13)$$

Figure 3. Example of a network scenario.

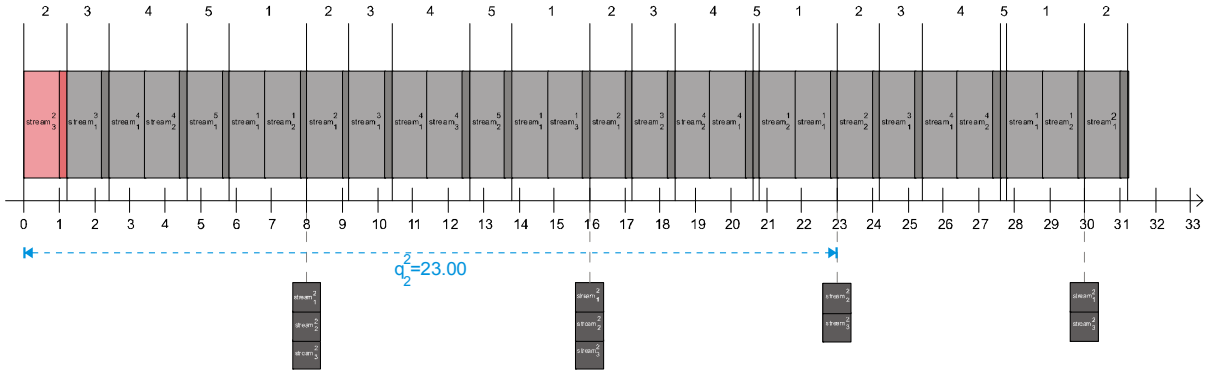

Figure 4. Network schedule for demonstration scenario.

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_{PR}. At the end of the node's turn, the time is increased by T_{MS}. Algorithm 4 illustrates this procedure.

3.5. Example 1

Let us put forward a demonstration scenario that will enable us to better grasp the algorithm behavior. Equations 12 and 13 in Figure 3 describe this demonstration scenario.

Figure 4 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 behavior. 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 5, when the activation time to be tested for S_2^2 is 0, which is the

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

Node	*Stream*	*Q* (time units)	*q* (time units)
$node^1$	S_1^1	8	8
	S_2^1	16	9
	S_3^1	16	16
	S_4^1	45	40
$node^2$	S_1^2	8	8
	S_2^2	24	23
	S_3^2	57	35
$node^3$	S_1^3	8	8
	S_2^3	45	32
$node^4$	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
$node^5$	S_1^5	8	8
	S_2^5	15	15

$$network = (3, \{node^1, node^2, node^3\}, 1, 1/5) \quad \textbf{(14)}$$

$$\begin{cases} node^1 = (3, \{S_1^1, S_2^1, S_3^1\}, \phi, RM, 1) \\ \quad S_1^1 = (5,5,0) \\ \quad S_2^1 = (13,13,0) \\ \quad S_3^1 = (13.4,13.4,0) \end{cases} \begin{cases} node^2 = (1, \{S_1^2\}, \phi, RM, 1) \\ \quad S_1^2 = (5.2,5.2,0) \end{cases} \quad \textbf{(15)}$$

$$\begin{cases} node^3 = (1, \{S_1^3\}, \phi, RM, 1) \\ \quad S_1^3 = (7,7,0) \end{cases}$$

Figure 5. Example of a network scenario (2).

Table 2. Queuing times for demonstration scenario (2).

Node	*Stream*	*Deadline* (time units)	*q using RM* (time units)	*q using EDF* (time units)
$node^1$	S_1^1	5	3.6	3.6
	S_2^1	13	10.4	5.2
	S_3^1	13.4	18.6	7.6
$node^2$	S_1^2	5.2	2.4	2.4
$node^3$	S_1^3	7	2.4	2.4

activation time leading to the maximum queuing delay.

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 labeled *q*). The column labeled *Q* contains the calculated queuing times using the analysis in [4]. We can see that our algorithm presents tighter results than the previous analysis.

3.6. Example 2

We will now compare the results given by the algorithm when RM or EDF scheduling policies are used. For this, we will the example network scenario given by Equations 14 and 15 in Figure 5.

The resulting queuing delays when using RM or EDF scheduling policies are given in Table 2.

We can see that using RM scheduling policy for the output queues of the nodes results that message stream S_3^1 loses its deadline. However, when using EDF, all deadlines are met.

4. Summary and Conclusions

We have presented an algorithm that computes exact queuing times for TDMA/SS in conjunction with Rate-Monotonic (RM) or Earliest-Deadline-First (EDF).

The algorithm was based on simulation by starting at the critical instant (for RM) or for many instants (for EDF). 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 [11] analysis for TDMA/SS and in that way achieve a polynomial time-complexity.

5. Acknowledgement

This work was partially funded by the Portuguese Science and Technology Foundation (FCT) and the ARTIST2 Network of Excellence on Embedded Systems Design.

6. References

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

[2] 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, issue 7, pp. 814 -826, 1996.

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

[4] B. Andersson, E. Tovar, and N. Pereira, "Analysing TDMA with Slot Skipping", In proc. of the 26th IEEE Real-time Systems Symposium (RTSS'05), Miami, FL, USA, 2005.

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

[6] 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, issue 3, pp. 229-249, 2002.

[7] L. George, N. Rivierre, and M. Spuri, "Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling", INRIA, Technical Report RR-2966, September 1996, online at: ftp://ftp.inria.fr/INRIA/publication/publi-pdf/RR/RR-2966.pdf.

[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 Syst., vol. 35, issue 3, pp. 239-272, 2007.

[9] S. K. Baruah, R. Howell, and L. Rosier, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor", Real-Time Systems, issue 2, pp. 301-324, 1990.

[10] B. Andersson, E. Tovar, and N. Pereira, "Analysing TDMA with Slot Skipping", Polytechnic Institute of Porto, Porto, Technical Report HURRAY-TR-050501, May 2005, online at: www.hurray.isep.ipp.pt.

[11] S. Chakraborty, S. Künzli, and L. Thiele, "Approximate Schedulability Analysis", In proc. of the IEEE Real-time Systems Symposium, Austin, Texas., 2002.