Timing Analysis of the FlexRay Communication Protocol

Traian Pop, Paul Pop, Petru Eles, Zebo Peng, Alexandru Andrei
Computer and Information Science Dept., Linköping University, Sweden
E-mail: {trapo, paupo, petel, zebpe, alean}@ida.liu.se

Abstract

FlexRay will very likely become the de-facto standard for in-vehicle communications. However, before it can be successfully used for safety-critical applications that require predictability, timing analysis techniques are necessary for providing bounds for the message communication times. In this paper, we propose techniques for determining the timing properties of messages transmitted in both the static (ST) and the dynamic (DYN) segments of a FlexRay communication cycle. The analysis techniques for messages are integrated in the context of a holistic schedulability analysis that computes the worst-case response times of all the tasks and messages in the system. We have evaluated the proposed analysis techniques using extensive experiments.

1. Introduction

Many safety-critical applications, following physical, modularity or safety constraints, are implemented using distributed architectures composed of several different types of hardware units (called nodes), interconnected in a network. For such systems, the communication between functions implemented on different nodes has an important impact on the overall system properties, such as performance, cost and maintainability.

There are several communication protocols for real-time networks. Among the protocols that have been proposed for in-vehicle communication, only the Controller Area Network (CAN) [4], the Local Interconnection Network (LIN) [17], and SAE's J1850 [30] are currently in use on a large scale [20]. Moreover, only a few of the proposed protocols are suitable for safety-critical applications where predictability is mandatory [29].

Communication activities can be triggered either dynamically, in response to an event (event-driven), or statically, at predetermined moments in time (time-driven). Therefore, on one hand, there are protocols that schedule the messages statically based on the progression of time, such as the SAFEbus [13], SPIDER [19], TTCAN [14], and Time-Triggered Protocol (TTP) [16]. The main drawback of such protocols is their lack of flexibility. On the other hand, there are communication protocols where message scheduling is performed dynamically, such as Byteflight [3] introduced by BMW for automotive applications, CAN [4], LonWorks [9] and Profibus [28].

A large consortium of automotive manufacturers and suppliers has recently proposed a hybrid type of protocol, namely the FlexRay communication protocol [11]. FlexRay allows the sharing of the bus among event-driven (ET) and time-driven (TT) messages, thus offering the advantages of both worlds. FlexRay will very likely become the de-facto standard for in-vehicle communications.[1] However, before it can be successfully deployed in applications that require predictability, timing analysis techniques are necessary to provide bounds for the message communication times [20].

FlexRay is composed of static (ST) and dynamic (DYN) segments, which are arranged to form a bus cycle that is repeated periodically. The ST segment is similar to TTP, and employs a generalized time-division multiple-access (GTDMA) scheme. The DYN segment of the FlexRay protocol is similar to Byteflight and uses a flexible TDMA (FTDMA) bus access scheme.

Although researchers have proposed analysis techniques for dynamic protocols such as CAN [32], TDMA [33], ATM [10], Token Ring protocol [31], FDDI protocol [1] and TTP [24], none of these analyses is applicable to the DYN segment in FlexRay. In [7], the authors consider the case of a hard real-time application implemented on a FlexRay bus. However, in their discussion they restrict themselves exclusively to the static segment, which means that, in fact, only the classical problem of communication scheduling over a TDMA bus [24, 12] is considered. The performance analysis of the Byteflight protocol, which is similar to the DYN segment of FlexRay, is analyzed in [5]. The authors assume a very restrictive "quasi-TDMA" transmission scheme for time-critical messages, which basically means that the DYN segment would behave as an ST (TDMA) segment in order to guarantee timeliness.

In this paper we present the first approach to timing analysis of applications communicating over a FlexRay bus, taking into consideration the specific aspects of this protocol, including the DYN segment. More exactly, we propose techniques for determining the timing properties of messages transmitted in the static and the dynamic segments of a FlexRay communication cycle. We first briefly present a static cyclic scheduling technique for TT messages transmitted in the ST segment, which extends our previous work on the TTP [23]. Then, we develop a worst-case response time analysis for ET messages sent using the DYN segment, thus providing predictability for messages transmitted in this segment. The analysis techniques for messages are integrated in the context of a holistic schedu-

1. Similar protocols exist in other industry areas. See WorldFIP [34] or MVB [15] for example.

Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS'06)
0-7695-2619-5 /06 $20.00 © 2006 IEEE

lability analysis algorithm that computes the worst-case response times of all the tasks and messages in the system.

This paper is organized in eight sections. Section 2 presents the system architecture considered, and Section 3 introduces the FlexRay media access control. In Section 4 we present the application model that we use. The main part of the paper is concentrated in Section 5, where we present our timing analysis for distributed real-time systems that use the FlexRay protocol. Section 6 extends the analysis to capture the independent usage of the two FlexRay channels. Section 7 presents the experimental results we have run in order to determine the efficiency of our approaches. The last section presents our conclusions.

2. System Model

We consider architectures consisting of nodes connected by one FlexRay communication channel[1] (see Figure 1.a). Each processing node connected to a FlexRay bus is composed of two main components: a CPU and a communication controller (see Figure 2.a) that are interconnected through a two-way controller-host interface (CHI). The controller runs independently of the node's CPU and implements the FlexRay protocol services.

For the systems we are studying, we have designed a software architecture which runs on the CPU of each node. The main component of the software architecture is a real-time kernel that contains two schedulers[2], for static cyclic scheduling (SCS) and fixed priority scheduling (FPS), respectively.

When several tasks are ready on a node, the task with the highest priority is activated, and preempts the other tasks. Let us consider the example in Figure 1.b, where we have six tasks sharing the same node. Tasks τ_1 and τ_6 are scheduled using SCS, while the rest are scheduled with FPS. The priorities of the FPS tasks are indicated in the figure. The arrival time of a task is depicted with an upwards pointing arrow. Under these assumptions, Figure 1.b presents the worst-case response times of each task. SCS tasks are non preemptable and their start time is off-line fixed in the schedule table (they also have the highest priority, denoted with priority level "0" in the figure). FPS tasks can only be executed in the slack of the SCS schedule table.

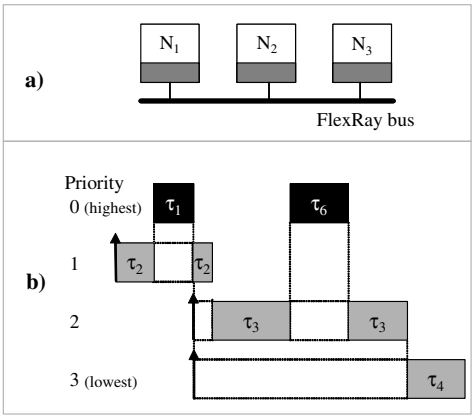

Figure 1. System Architecture Example

1. A dual-channel FlexRay bus is considered in Section 6
2. EDF can also be added, as presented by us in [27]

FPS tasks are scheduled based on priorities. Thus, a higher priority task such as τ_3 preempts a lower priority task such as τ_4. SCS activities are triggered based on a local clock in each processing node. The synchronization of local clocks throughout the system is provided by the communication protocol [11].

3. The FlexRay Communication Protocol

In this section we will describe how messages generated by the CPU reach the communication controller and how they are transmitted on the bus. Let us consider the example in Figure 2 where we have three nodes, N_1 to N_3 sending messages m_a, m_b,... m_h using a FlexRay bus.

In FlexRay, the communication takes place in periodic cycles (Figure 2.b depicts two cycles of length T_{bus}). Each cycle contains two time intervals with different bus access policies: an ST segment and a DYN segment[3]. The ST and DYN segment lengths can differ, but are fixed over the cycles. We denote with ST_{bus} and DYN_{bus} the length of these segments. Both the ST and DYN segments are composed of several slots. In the ST segment, the slots number is fixed, and the slots have constant and equal length, regardless of whether ST messages are sent or not over the bus in that cycle. The length of an ST slot is specified by the FlexRay global configuration parameter *gdStaticSlot* [11]. In Figure 2 there are three static slots for the ST segment.

The length of the DYN segment is specified in number of "minislots", and is equal to *gNumberOfMinislots.* Thus, during the DYN segment, if no message is to be sent during a certain slot, then that slot will have a very small length (equal to the length *gdMinislot* of a so called minislot), otherwise the DYN slot will have a length equal with the number of minislots needed for transmitting the whole message [11]. This can be seen in Figure 2.b, where DYN slot 2 has 3 minislots (4, 5, and 6) in the first bus cycle, when message m_e is transmitted, and one minislot (denoted with "MS" and corresponding to the minislot counter 2) in the second bus cycle when no message is sent.

During any slot (ST or DYN), only one node is allowed to send on the bus, and that is the node which holds the message with the frame identifier (*FrameID*) equal to the current value of the slot counter. There are two slot counters, corresponding to the ST and DYN segments, respectively. The assignment of frame identifiers to nodes is static and decided offline, during the design phase. Each node that sends mes-

3. The FlexRay bus cycle contains also a *symbol window* and a *network idle time*, but their size does not affect the equations in our analysis. For simplicity, they will be ignored during the examples throughout the paper.

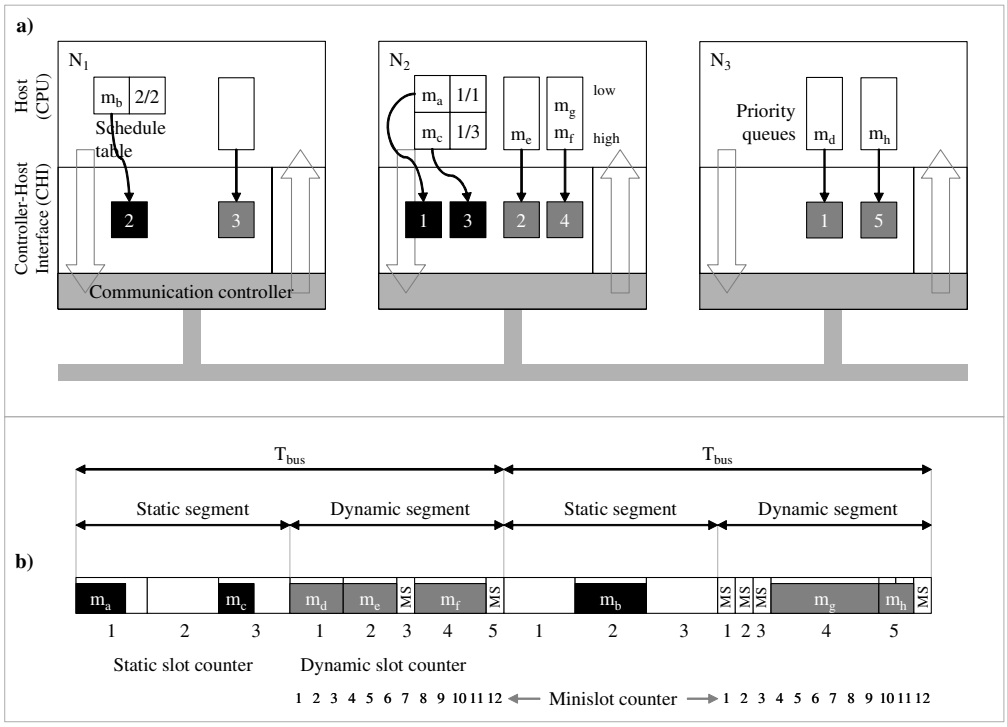

Figure 2. FlexRay Communication Cycle Example

sages has one or more ST and/or DYN slots associated to it. The bus conflicts are solved by allocating offline one slot to at most one node, thus making it impossible for two nodes to send during the same ST or DYN slot.

In Figure 2, node N_1 has been allocated ST slot 2 and DYN slot 3, N_2 transmits through ST slots 1 and 3 and DYN slots 2 and 4, while node N_3 has DYN slots 1 and 5. For each of these slots, the CHI reserves a buffer that can be written by the CPU and read by the communication controller (these buffers are read by the communication controller *at the beginning* of each bus cycle, in order to prepare the transmission of frames) The associated buffers in the CHI are depicted in Figure 2.a. We denote with $DYNSlots_{N_p}$ the number of dynamic slots associated to a node N_p (this means that for N_2 in Figure 2, $DYNSlots_{N_2} = 2$).

We use different approaches for ST and DYN messages to decide which messages are transmitted during the allocated slots. For ST messages, we consider that the CPU in each node holds a schedule table with the transmission times. When the time comes for an ST message to be transmitted, the CPU will place that message in its associated ST buffer of the CHI. For example, ST message m_b sent from node N_1 has an entry "2/2" in the schedule table specifying that it should be sent in the second slot of the second ST cycle.

For the DYN messages, the designer specifies their *FrameID*. For example, DYN message m_e has the frame identifier "2". We assume that there can be several messages sharing the same DYN *FrameID*[1]. For example, messages m_g and m_f have both *FrameID* 4. If two messages with the same frame identifier are ready to be sent in the same bus cycle, a priority scheme is used to decide which message will be sent first. Each DYN message m_i has associated a priority $priority_{m_i}$. Messages with the same *FrameID* will be placed in an output queue ordered based on their priorities. The message form the head of the priority queue is sent in the current bus cycle. For example, message m_f will be sent before m_g because it has a higher priority.

At the beginning of each communication cycle, the communication controller of a node resets the slot and minislot counters. At the beginning of each communication slot, the controller verifies if there are messages ready for transmission (present in the CHI send buffers) and packs them into frames[2]. In the example in Figure 2 we assume that all messages are ready for transmission before the first bus cycle.

Messages selected and packed into ST frames will be transmitted during the bus cycle that is about to start according to the schedule table. For example, in Figure 2, messages m_a and m_c are placed into the associated ST buffers in the CHI in order to be transmitted in the first bus cycle. However, messages selected and packed into DYN frames will be transmitted during the DYN segment of the

1. This assumption is not part of the FlexRay specification. If messages are not sharing FrameIDs, this is handled implicitly as a particular case of our analysis.
2. In this paper we do not address frame-packing [25], and thus assume that one message is sent per frame.

bus cycle only if there is enough time until the end of the DYN segment. Such a situation is verified by comparing if, in the moment the DYN slot counter reaches the value of the *FrameID* for that message, the value of the minislot counter is smaller than a given value *pLatestTx*. The value *pLatestTx* is fixed for each node during the design phase, depending on the size of the largest DYN frame that node will have to send during run-time. For example, in Figure 2, message m_h is ready for transmission before the first bus cycle starts, but, after message m_f is transmitted, there is not enough room left in the DYN segment. This will delay the transmission of m_h for the next bus cycle.

4. Application Model

We model an application $\mathcal{A}$ as a set of directed, acyclic, polar graphs $\mathcal{G}_i(\mathcal{V}_i, \mathcal{E}_i) \in \mathcal{A}$. A node $\tau_{ij} \in \mathcal{V}_i$ represents the *j*-th task or message in $\mathcal{G}_i$. An edge $e_{ijk} \in \mathcal{E}_i$ from τ_{ij} to τ_{ik} indicates that the output of τ_{ij} is the input of τ_{ik}. A task becomes ready after all its inputs have arrived and it issues its outputs when it terminates. A message will become ready after its sender task has finished, and becomes available for the receiver task after its transmission has ended. The communication time between tasks mapped on the same processor is considered to be part of the task worst-case execution time and is not modeled explicitly. Communication between tasks mapped to different processors is performed by message passing over the bus. Such message passing is modeled as a communication task inserted on the arc connecting the sender and the receiver task.

We consider that the scheduling policy for each task is known (either *SCS* or *FPS*), and we also know which messages are ST and which are DYN. For a task $\tau_{ij} \in \mathcal{V}_i$, $Node_{\tau_{ij}}$ is the node to which τ_{ij} is assigned for execution. When executed on $Node_{\tau_{ij}}$, a task τ_{ij} has a known worst-case execution time $C_{\tau_{ij}}$. We also consider that the size of each message *m* is given, which can be directly converted into communication time C_m on the particular bus, knowing the speed of the bus and the size of the frame that stores the message:

$$C_m = Frame\_size(m) / bus\_speed. \quad (1)$$

Tasks and messages activated based on events also have a priority, $priority_{\tau_{ij}}$. All tasks and messages belonging to a task graph $\mathcal{G}_i$ have the same period $T_{\tau_{ij}} = T_{\mathcal{G}_i}$ which is the period of the process graph. A deadline $D_{\mathcal{G}_i}$ is imposed on each task graph $\mathcal{G}_i$. In addition, tasks can have associated individual release times and deadlines. If communicating tasks are of different periods, they are combined into a larger graph capturing all task activations for the hyper-period (LCM of periods).

5. Timing Analysis

Given a distributed system based on FlexRay, as described in the previous two sections, the tasks and messages have to be scheduled. For the SCS tasks and ST messages, this means building the schedule tables, while for the FPS tasks and DYN messages we have to determine their worst case response times.

The problem of finding a schedulable system has to consider two aspects:

1. When performing the schedulability analysis for the FPS tasks and DYN messages, one has to take into consideration the interference from the SCS activities.
2. Among the possible correct schedules for SCS activities, it is important to build one which favours as much as possible the schedulability of FPS activities.

Figure 3 presents the global scheduling and analysis algorithm, in which the main loop consists of a list-scheduling based algorithm [6] that iteratively builds the static schedule table with start times for SCS tasks and ST messages.

A ready list (*TT_ready_list*) contains all SCS tasks and ST messages which are ready to be scheduled (they have no predecessors or all their predecessors have already been scheduled). From the ready list, tasks and messages are extracted one by one (Figure 3, line 2) to be scheduled on the processor they are mapped to (line 4), or into a static bus-slot associated to that processor on which the sender of the message is executed (line 6), respectively. The priority function which is used to select among ready tasks and messages is a critical path metric, modified by us for the particular goal of scheduling tasks mapped on distributed systems [23]. Let us consider a particular task τ_{ij} selected from the ready list to be scheduled. We consider that $ASAP_{\tau_{ij}}$ is the earliest time moment which satisfies the condition that all preceding activities (tasks or messages) of τ_{ij} are finished (line 10). With only the SCS tasks in the

```
GlobalSchedulingAlgorithm()
1   while TT_ready_list is not empty
2       select τij from TT_ready_list
3       if τij is a SCS task then
4           schedule_TT_task(τij, Node_τij)
5       else // τij is a ST message
6           schedule_ST_msg(τij, Node_τij)
7       end if
8       update TT_ready_list
9   end while
end StaticScheduling

schedule_TT_task(τij, Node_τij)
10      find first available time t moment after ASAP_τij on Node_τij
11      schedule τij after t on Node_τij so that holistic analysis produces
        minimal worst-case response times for FPS tasks and DYN messages
12      update ASAP for all τij successors
end schedule_TT_task
schedule_ST_msg(τij, Node_τij)
13      find first ST slot(Node_τij) available after ASAP_τij
14      schedule τij in that ST slot
15      update ASAP for all τij successors
end schedule_ST_msg
```

Figure 3. Global Scheduling Algorithm

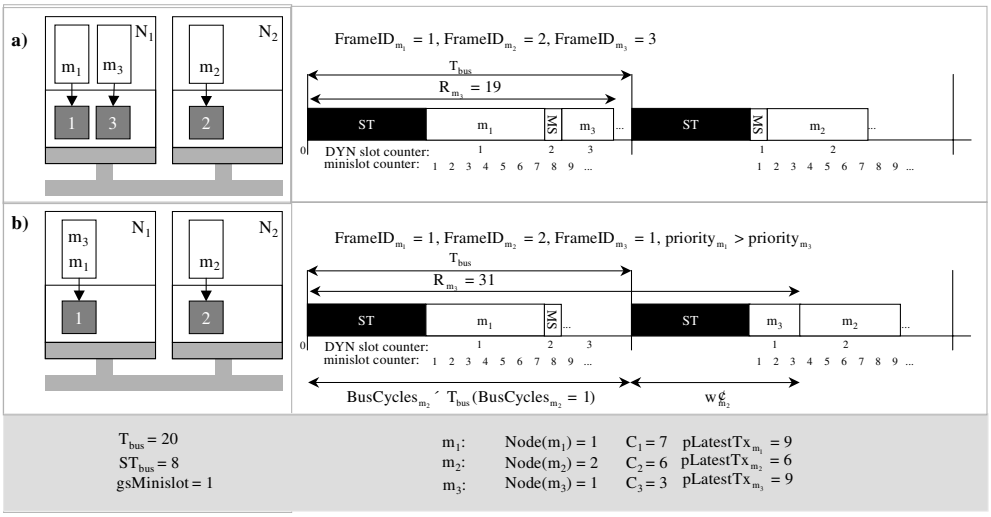

Figure 4. Transmission Scenarios for DYN Messages

system, the straightforward solution would be to schedule τ_{ij} at the first time moment after $ASAP_{\tau_{ij}}$ when $Node_{\tau_{ij}}$ is free. Similarly, an ST message will be scheduled in the first available ST slot associated with the node that runs the sender task for that message.

As presented by us in [26], when scheduling SCS tasks, one has to take into account the interference they produce on FPS tasks. The function *schedule_TT_task* in Figure 3 places a SCS task in the static schedule in such a way that the increase of worst-case response times for FPS tasks is minimized. Such an increase is determined by comparing the worst-case response times of FPS tasks obtained with our holistic schedulability analysis before and after inserting the new SCS task in the schedule [26].

The next subsection presents our solution for computing the worst case response times of DYN messages, while in Section 5.2 we will integrate this solution into a holistic schedulability analysis that determines the timing properties of both FPS tasks and DYN messages (which is called in line 11, of *schedule_TT_task* presented in Figure 3).

5.1 Schedulability Analysis of DYN Messages

The worst case response time R_m of a DYN message m is given by the following equation:

$$R_m(t) = \sigma_m + w_m(t) + C_m \quad (2)$$

where C_m is the message communication time (see Section 4), σ_m is the longest delay suffered during one bus cycle if the message is generated by its sender task after its slot has passed, and w_m is the worst case delay caused by the transmission of ST frames and higher priority DYN messages during a given time interval t.

The communication controller decides what message is to be sent on the bus in a certain communication slot *at the beginning* of that slot. As a consequence, in the worst case, a DYN message m is generated by its sender task immediately after the slot with the $FrameID_m$ has started, forcing message m to wait until the next bus cycle starts in order to really start competing for the bus. In conclusion, in the worst case, the delay σ_m has the value:

$$\sigma_m = T_{bus} - (ST_{bus} + FrameIDm \cdot gdMinislot) \quad ,$$

where ST_{bus} is the length of the ST segment.

What is now left to be determined is the value w_m corresponding to the maximum amount of delay on the bus that can be produced by interference from ST frames and higher priority DYN messages. We start from the observations that the transmission of a ready DYN message m during the DYN slot $FrameID_m$ can be delayed because of the following causes:

- local messages with higher priority, that use the same frame identifier as m. We will denote this set of *higher priority local messages* with $hp(m)$. For example, in Figure 2.a, messages m_g and m_f share *FrameID* 4, thus $hp(m_g) = \{m_f\}$.
- any messages in the system that can use DYN slots with lower frame identifiers than the one used by m. We will denote this set of messages having *lower frame identifiers* with $lf(m)$. In Figure 2.a, $lf(m_g) = \{m_d, m_e\}$.
- unused DYN slots with frame identifiers lower than the one used for sending m (though such slots are unused, each of them still delays the transmission of m for an interval of time equal with the length *gdMinislot* of one minislot); we will denote the set of such minislots with $ms(m)$. Thus, in the example in Figure 2.a, $ms(m_g) = \{1, 2, 3\}$, and $ms(m_f) = \{3\}$.

Determining the interference of DYN messages in FlexRay is complicated by several factors. Let us consider the example in Figure 4, where we have two nodes, N_1 (with *FrameIDs* 1 and 3) and N_2 (with *FrameID* 2), and three messages m_1 to m_3. N_1 sends m_1 and m_3, and N_2 sends mes-

sage m_2. Messages m_1 and m_2 have *FrameIDs* 1 and 2, respectively. We consider two situations: Figure 4.a, where m_3 has a separate *FrameID* 3, and Figure 4.b, where m_3 shares the same *FrameID* 1 with m_1. The values of *pLatestTx* for each node are depicted in the figure[1].

In Figure 4.a, message m_2 cannot be sent immediately after message m_1, because the value of the minislot counter has exceeded the value $pLatestTx_{m_2}$ when the value of the DYN slot counter becomes equal to 2. As a consequence, the transmission of m_2 will be delayed for the next bus cycle. However, since in the moment when the DYN slot counter becomes 3 the minislot counter does not exceed the value $pLatestTx_{m_3}$, message m_3 will fit in the first bus cycle. Thus, a message (m_3 in our case) can be sent before another message with a lower *FrameID* (m_2). Such situations must be accounted for when building the worst-case scenario.

In Figure 4.b, message m_3 shares the same *FrameID* 1 with m_1 but we consider that it has a lower priority, thus $hp(m_3) = \{m_1\}$. In this case, m_3 is sent in the first DYN slot of the second bus cycle (the first slot of the first cycle is occupied with m_1) and thus will delay the transmission of m_2. In this scenario, we notice that assigning a lower frame identifier to a message does not necessarily reduce the worst-case response time of that message (compare to the situation in Figure 4.a, where m_3 has *FrameID* = 3).

We next focus on determining the delay $w_m(t)$ in Equation (2). The delay produced by all the elements in $hp(m)$, $lf(m)$ and $ms(m)$ can extend to one or more bus cycles. As a consequence, Equation (2) for finding the worst case response time R_m can be rewritten as:

$$R_m(t) = \sigma_m + BusCycles_m(t) \times T_{bus} + w'_m(t) + C_m \quad (3)$$

where $BusCycles_m(t)$ is the number of bus periods for which the transmission of m is not possible because transmission of messages from $hp(m)$ and $lf(m)$ and because of minislots in $ms(m)$. The delay $w'_m(t)$ denotes now the time, in the last bus cycle, until m is sent, and is measured from the beginning of the bus cycle in which message m is sent until the actual transmission of m starts. For example, in Figure 4.b, $BusCycles_{m_2} = 1$ and $w'_{m_2}(t) = ST_{bus} + C_{m_3}$. Note that both these terms are functions of time, computed over an analyzed interval t. This means that when computing them we have to take into consideration all the elements in $hp(m)$, $lp(m)$ and $ms(m)$ that can appear during such a given time interval t. Thus, we will consider the multiset $hp(m, t)$ containing all the occurrences over t of elements in $hp(m)$. The number of such occurrences for a message $l \in hp(m)$ is equal to: $\lceil (J_l + t)/T_l \rceil$, where T_l is the period of the message l and J_l is its worst-case jitter (such a jitter is computed as the difference between the worst-case and best-case response times of its sender task s: $J_l = R_s - R_s^b$ [21]). Similarly, $lf(m, t)$ and $ms(m, t)$ consider all the occurrences over t of elements in $lf(m)$ and $ms(m)$ respectively.

The next two sections (5.1.1 and 5.1.2) present the optimal (i.e., exact) solutions for determining the values for $BusCycles_m(t)$ and $w'_m(t)$, respectively. These, however, can be intractable for larger problem sizes. Hence, in Sections 5.1.3 and 5.1.4 we propose heuristics that quickly computer upper bounds (i.e., pessimistic) values for these terms. Once for any given t we know how to obtain the values $BusCycles(t)$ and $w'_m(t)$, determining the worst case response time for a message m becomes an iterative process that computes $R_m^k(R_m^{k-1})$, starting from $R_m^0 = C_m$ and finishing when $R_m^k = R_m^{k-1}$.

5.1.1 Optimal Solution for $BusCycles_m$

We start with the observation that a message m with $FrameID_m$ cannot be sent by a node N_p during a bus cycle b if at least one of the following conditions is fulfilled:

1. There is too much interference from elements in $lf(m)$ and $ms(m)$, so that the minislot counter exceeds the value $pLatestTx_{N_p}$, making impossible for N_p to start the transmission of m during b. For example in Figure 4.a, message m_2 cannot be sent during the first bus cycle because the transmission of a higher priority message m_1 pushes the minislot counter over the value $pLatestTx_{N_1}$.
2. The DYN slot $FrameID_m$ in b is used by another local higher priority message from $hp(m)$. For example, in Figure 4.b, messages m_1 and m_3 share the same frame identifier and $hp(m_3) = \{m_1\}$. Therefore, the transmission of m_3 in the first bus cycle is not possible.

Whenever a bus cycle satisfies at least one of these two conditions, it will be called "filled", since it is unusable for the transmission of the message m under analysis. In the worst case, the value $BusCycles_m(t)$ is then the maximum number of bus cycles that can be filled using elements from $hp(m)$, $lf(m)$ and $ms(m)$.

1. We use $pLatestTx_m$ to denote $pLatestTx_N$ of the node N sending message m.

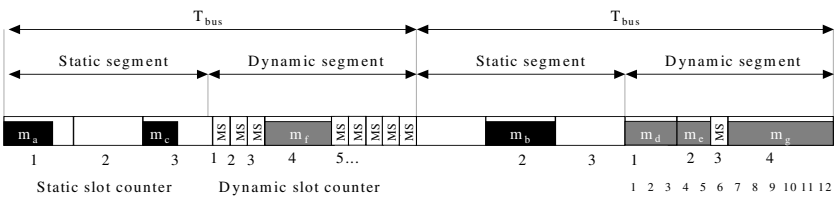

Figure 5. Worst Case Scenario for DYN frames

Since messages in $hp(m, t)$ and $lf(m, t)$ can become ready at any point during the analyzed interval t, one can notice that, in the worst case, each bus cycle which is filled with an element from $hp(m, t)$ will contain no messages from $lf(m, t)$. This means that in the worst case, each filled bus cycle will contain either only messages from $lf(m, t)$, or only one message from $hp(m, t)$. For example, considering the same setup presented in Figure 2, the worst-case scenario for message m_g is when message m_f is ready at the beginning of the first bus cycle and messages m_d and m_e become ready just before the start of their slots in the second bus cycle (see Figure 5 for the worst-case scenario of m_g).

This means that, in the worst case, the delay produced by elements in $lf(m, t)$ and $ms(m, t)$ adds up to that produced by messages in $hp(m, t)$:

$$BusCycles_m(t) = BusCycles_m(hp(m, t)) + BusCycles_m(lf(m, t), ms(m, t)) \quad (4)$$

where we denote with $BusCycles_m(hp(m, t))$ the number of bus cycles in which the delay of the message m under analysis is produced by messages in $hp(m, t)$ (corresponding to the second case presented above); similarly, $BusCycles_m(lf(m, t), ms(m, t))$ is the number of "filled" bus cycles in which the transmission of message m is delayed by elements in $lf(m, t)$ and $ms(m, t)$ (corresponding to the first condition presented above).

Since each message in $hp(m, t)$ delays the transmission of m with one bus cycle, the occurrences over t of messages in $hp(m)$ will produce a delay equal to the total number of elements in $hp(m, t)$:

$$BusCycles_m(hp(m, t)) = |hp(m, t)| \ . \quad (5)$$

The problem that remains to be solved is to determine how many bus cycles can be "filled" according to the first condition presented above using only elements in $lf(m, t)$ and $ms(m, t)$. As we will discuss later, a simplified version of this problem is equivalent to bin covering, which belongs to the family of NP-hard problems [8]. To obtain the optimal solution, we have modelled the problem of computing $BusCycles_m(lf(m, t), ms(m, t))$ as an integer linear program (ILP). The model starts from the observation that, considering we have n elements in $lf(m, t)$, there are at most n bus cycles that can be filled. For each such bus cycle we create a binary variable $y_{i=1..n}$ that is set to 1 when the i-th bus cycle is filled with elements from $lf(m, t)$ and $ms(m, t)$, and to 0 if it is not filled (i.e., it can allow the transmission of message m under analysis).

The goal of the ILP problem is to maximize the number of filled bus cycles (i.e., to calculate the worst-case):

$$BusCycles_m(lf(m, t), ms(m, t)) = \sum_{i = 1..n} y_i \ , \quad (6)$$

subject to a set of conditions that set the variables y_i to 1 or 0. Bellow we describe these conditions, which capture how messages in $lf(m, t)$ and the minislots in $ms(m, t)$ are sent by FlexRay in these bus cycles.

We allocate a binary variable x_{ijk} that is set to 1 if a message $m_k \in lf(m, t)$ $(k = 1..n)$ is sent during the i-th bus cycle, using the $FrameID\ j = 1..FrameID_m$. The load transmitted in each bus cycle can be expressed as:

$$Load_i = \sum_{\substack{m_k \in lf(m, t) \\ j = 1 \ldots FrameID_m}} x_{ijk} \times C_k + \sum_{j = 1 \ldots FrameID_m} \left(1 - \sum_{m_k \in lf(m, t)} x_{ijk}\right) \times gdMinislot \quad (7)$$

where C_k are the communication times (Equation (1)) of the messages $m_k \in lf(m, t)$. Each term of the sum in Equation (7) captures the particularities of FlexRay DYN frames: if a message k is transmitted in cycle i with frame identifier j, then $x_{ijk} = 1$ and the length of the frame being transmitted is equal with the length of the message k, (thus the term $x_{ijk} \times C_k$); if x_{ijk} is 0 for all j and k, then there is no actual transmission on the bus in that DYN slot, but there is still some delay due to the empty minislot of length *gdMinislot* that has to pass in order to increase the value of the DYN slot counter (thus the second term).

The condition that sets each variable y_i to 1 whenever possible is:

$$Load_i > pLatestTx_{N_p} \times gdMinislot \times y_i \quad (8)$$

where $pLatestTx_{N_p}$ is the last minislot which allows the start of transmission from node N_p of message m under analysis. Such a condition enforces that a variable y_i cannot be set to 1 unless the total amount of interference from $lf(m, t)$ and $ms(m, t)$ in cycle i exceeds $pLatestTx_{N_p}$ minislots (only then message m is not allowed to be transmitted and, thus, bus cycle i is "filled").

In addition to this condition we have to make sure that

- each message $m_k \in lf(m, t)$ is sent in only one cycle i:

$$\sum_{\substack{i = 1 \ldots n \\ j = 1 \ldots FrameID_m}} x_{ijk} \leq 1, \forall m_k \in lf(m) \quad (9)$$

$$(10)$$

- each frame identifier is used only once in a bus cycle:

$$\sum_{k = 1 \ldots n} x_{ijk} \leq 1, \forall i, j \quad (11)$$

- each message $m_k \in lf(m, t)$ is transmitted using its frame identifier:

$$x_{ijk} \leq Frame_{jk}, \ \forall i, j, k \quad (12)$$

where $Frame_{jk}$ is a binary constant with value 1 if message $m_k \in lf(m, t)$ has a frame identifier $FrameID_{m_k} = j$ (otherwise, $Frame_{jk}$ is 0).

Finally, we have to enforce that in every cycle i no message m_k will start transmission after its associated $pLatestTx_{m_k}$. If we have $x_{ijk} = 1$, then we have to add the condition that the total amount of transmission that takes

place before DYN slot j has to finish no later than $pLatestTx_k$:

$$\sum_{\substack{m_q \in lf(m,t) \\ p = 1..j-1}} x_{ipq} \times C_q + \sum_{p = 1..j-1} \left(1 - \sum_{m_q \in lf(m,t)} x_{ipq}\right) \times gdMinislot \leq pLatestTx_k \times gdMinislot \quad (13)$$

The conditions (7)–(13) together with the maximization goal expressed in Equation (6) define the ILP program that will determine the maximum worst-case number of bus cycles that can be filled with elements in $lf(m, t)$ and $ms(m, t)$. By adding this result to the value determined in Equation (5), we obtain the total number $BusCycles_m(t)$ (Equation (4)).

5.1.2 Optimal Solution for w'_m

In the worst case, the elements in $lf(m,t)$ and $ms(m,t)$ will delay the message under analysis for $BusCycles_m$ ($lf(m,t)$, $ms(m,t)$) bus periods. In addition, they will delay the actual transmission of m during the DYN segment of the bus period $BusCycles_m + 1$.

The problem of determining the value for w'_m is defined as follows: given the multisets $lf(m,t)$ and $ms(m,t)$ and the maximum number $BusCycles_m(lf(m,t), ms(m,t))$ that they can fill, what is the maximum possible load (Equation (7)) in the first unfilled bus cycle (i.e. the bus cycle that does not satisfy condition (8)).

In order to determine the exact value of w'_m in the worst case, one can use the same ILP system defined in the previous section for computing $BusCycles_m(lf(m,t), ms(m,t))$, with the following modifications:

- since we know the value $BusCycles_m$ (which is determined solving the ILP formulation presented in the previous section), we add conditions that force the values $y_i = 1$ for all i=1..$BusCycles_m$, and $y_i = 0$ for all $i = BusCycles_m + 1..n$; in this way, the messages will be packed so that the bus cycles from 1 to $BusCycles_m$ will be filled (i.e they satisfy condition (8)), while the remaining bus cycles will be unfilled.
- using the same set of conditions (7)–(13) for filling the first $BusCycles_m$ cycles, the goal described in Equation (6) is replaced with the following one, expressing that the load of the cycle number $BusCycles_m + 1$ has to be maximized ($Load_L$ is expressed as in Equation (7)):

$$maximize\ Load_L, \text{ for } L = BusCycles_m + 1 \quad (14)$$

5.1.3 Heuristic Solution for $BusCycles_m$

We first make the observation that in a bus cycle where a message m is sent by a node N_p during DYN slot $FrameID_m$, in the worst case there will be at most $FrameID_m - 1$ unused minislots before m is transmitted (In Figure 4.a, the transmission of m_2 can be preceded by at most one unused minislot).

Instead of considering the multiset $ms(m, t)$ as for the exact solution, we will account for the worst-case as part of the communication time for m:

$$C'_m = (FrameID_m - 1) \times gdMinislot + C_m. \quad (15)$$

Since the duration of one minislot ($gdMinislot$) is an order of magnitude smaller compared to the length of a cycle, this approximation will not introduce any significant pessimism.

The problem left to solve now is how many bus cycles can be filled with the elements from a multiset $lf'(m, t)$, that consists of all the messages in $lf(m, t)$ for which we consider the communication times computed using Equation (15).

If we ignore the conditions expressed in equations (11)–(13), then determining $BusCycles_m(lf'(m, t))$ becomes a *bin covering* problem [8]. Bin covering tries to maximize the number of bins that can be filled to a fixed minimum capacity using a given set of items with specified weights. In our scenario, the messages in $lf'(m, t)$ are the items, the dynamic segments of the bus cycles are bins, and $pLatestTx_{N_p} \times gdMinislot$ is the minimum capacity required to fill a bin. The bin-covering problem is NP-hard in the strong sense [8], and our solution is to determine an upper bound, using the approach presented in [8], on the number of maximum bins that can be covered. The upper bound proposed in [8] are of polynomial complexity and obtain very good quality results.

Note that, ignoring the conditions from (11)–(13) and determining an upper bound for bin-covering can only lead to an increase in the number of bus cycles compared to the exact solution. Experiments will show the impact of the heuristic on the pessimism of the analysis.

5.1.4 Heuristic Solution for w'_m

A straightforward heuristic to the computation of w'_m stems from the observation that, in a hypothetical worst-case scenario, message m could be sent in the last possible moment of the current bus cycle, which means that

$$w'_m = ST_{bus} + pLatestTx_{N_p} \times gdMinislot, \quad (16)$$

where ST_{bus} is the length of the ST segment of a bus cycle.

5.2 Holistic Schedulability Analysis of FPS Tasks and DYN Messages

As mentioned in Section 2, the worst-case response times of FPS tasks are influenced on one hand by higher priority FPS tasks, and on the other hand by SCS tasks. The worst-case response time R_{ij} of a FPS task τ_{ij} is determined as presented in [21], and in [26] we have shown how to take into consideration the interference on R_{ij} produced by an

existing static schedule. What is important to mention is that R_{ij} depends on jitters of the higher priority tasks and predecessors of τ_{ij}. This means that for all such activities we have to compute the jitter. In the rest of this section we will only concentrate on the situation when the jitter of a task depends on the arrival time of a message.

According to the analysis of multiprocessor and distributed systems presented in [21], the jitter for a task τ_r that starts execution only after it receives a message m depends on the values of the best-case and worst-case transmission times of that message:

$$J_{\tau_r} = R_m - R_m^b. \quad (17)$$

The calculation of the worst-case transmission time R_m of a DYN message m was presented in Section 5.1. For computing R_m^b we have to identify the best-case scenario of transmitting message m. Such a situation appears when the message becomes ready immediately before the DYN slot with $FrameID_m$ starts, and it is sent during that bus cycle without experiencing any delay from higher priority messages. Thus, the equation for the best-case transmission time of a message is:

$$R_m^b = C_m \quad (18)$$

where C_m is the time needed to send the message m.

We notice from Equation (17) that the jitters for activities in the system depend on the values of the worst case response times, which in turn depend on the values of the jitters [27]. Such a recursive system is solved using a fixed point iteration algorithm in which the initial values for jitters are 0.

Let us make a final remark. According to [21], the worst-case response time calculation of FPS tasks is of exponential complexity and the approach proposed in [21] and also used in [27] is a heuristic with a certain degree of pessimism. The pessimism of the response times calculated by our holistic analysis will, of course, also depend on the quality of the solution for the delay induced by the DYN messages transmitted over FlexRay. The calculation of this delay is our main concern in this paper. Therefore, when we speak about optimal and heuristic solutions in this paper we refer to the approach used for calculating the $BusCycles_m$ and w'_m (used in the worst-case response times calculation for DYN messages) and not the holistic response time analysis which is based on the heuristics in [21, 26].

6. Analysis for Dual-channel FlexRay Bus

The specification of the FlexRay protocol mentions that the bus has two communication channels [11]. The analysis presented in section 5 is appropriate for systems where the two channels of the FlexRay bus are used in a redundant manner, transporting the same information simultaneously in order to support fault-tolerance.

In order to increase the bandwidth of the bus, one can use the two channels independently, so that different sets of messages are sent over each of the channels during a bus cycle. In this section we extend our previous analysis in order to compute the worst case response times for messages transmitted in such systems.

First, we extend our system model (Section 1.a) and consider that all nodes in the system have access to a dual-channel FlexRay bus. As a consequence, in the application model each message m is associated a pair $<FrameID_m, Channel_m>$, with the meaning that message m is sent during $FrameID_m$ on $Channel_m$ (where $Channel_m = \{A, B\}$).

Second, we notice that the transmission of a message can be delayed only by messages that are transmitted on the same channel. As a consequence, the only modification in the analysis presented in section 5 is the definition of the sets $lf(m)$ and $hp(m)$, which contain only those messages that are transmitted on $Channel_m$:

- $hp(m)$ becomes now the set of local messages with higher priority, that use the same frame identifier AND the same channel as m.
- $lf(m)$ contains any messages in the system that can use $Channel_m$ and DYN slots with lower frame identifiers than the one used by m.

7. Experimental Results

We were interested to determine the quality of the proposed analysis approaches, and how well they scale with the number of FlexRay messages that have to be analyzed. All the experiments were run on P4 machines using 2GB RAM. The ILP-based solutions have been implemented using the CPLEX 9.1.2 ILP solver.

We have generated synthetic applications of 20, 30, 40 and 50 tasks mapped on architectures consisting of 2, 3, 4, and 5 nodes, respectively. Fifteen applications were generated for each of these four cases. The number of time-critical FlexRay messages were 30, 60, 90, and 120 for each case, respectively. Out of these, 10, 20, 30, and 40 messages were time-critical DYN messages that were analyzed using the approaches presented in Section 5. Each application has been analyzed using four holistic analysis approaches, depending on the approach used for the calculation of the components $BusCycles_m$ and w'_m of the worst-case response time R_m for a DYN message:

Holistic Analysis	$BusCycles_m$	w'_m
OO	**O**ptimal solution (5.1.1)	**O**ptimal solution (5.1.2)
OO$^-$	**O**ptimal solution (5.1.1)	ILP from 5.1.2 with 1 min. time-out (**O**$^-$)
OH	**O**ptimal solution (5.1.1)	**H**euristic solution (5.1.4)
HH	**H**euristic solution (5.1.3)	**H**euristic solution (5.1.4)

OO will always provide the tightest worst-case response times. However, it is only able to produce results for up to 20 DYN messages in a reasonable time. We have noticed that the bottleneck for OO is the exact calculation of w'_m (which is a value smaller than a bus cycle), and that

No of msgs.	30 (10 DYN)		60 (20 DYN)		90 (30 DYN)		120 (40 DYN)	
	Ratio	Exec. (s)	Ratio	Exec. (s)	Ratio	Exec. (s)	Ratio	Exec. (s)
OO	1.009	3.1 s	1.009	42.3 s	–	–	–	–
OH	1.013	1.29 s	1.012	14.42 s	1.005	57.32 s	1.005	367.87 s
HH	1.016	0.012 s	1.018	0.019 s	1.012	0.036 s	1.012	0.04 s

Table 1: Comparison of FlexRay Analysis Approaches

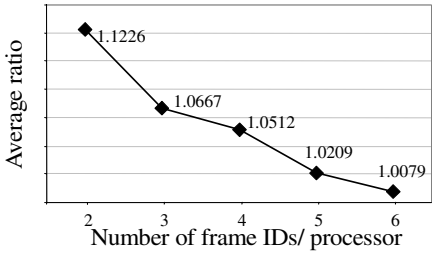

Figure 6. Quality of HH

running the ILP from Section 5.1.2 using a time-out of one minute we are able to obtain near-optimal results for w'_m. We have denoted with OO$^-$ such an analysis. Since the near-optimal result for w'_m is a lower bound, OO$^-$ can lead to an incorrect (optimistic) result (i.e., the system is reported as schedulable, but in reality it might not be). Although OO$^-$ is thus of no practical use, it is very useful in determining, by comparison, the quality of our proposed FlexRay analysis heuristics, OH and HH.

In order to evaluate the approaches for FlexRay analysis, we have determined for an analysis approach A the average ratio:

$$\text{ratio} = \frac{1}{n} \cdot \sum_{m \in DYN} \frac{R_m^A}{R_m^{\text{OO}^-}} \tag{19}$$

where A is one of the OO, OH or HH approaches and n is the number of messages in the analysed application.

This ratio captures the degree of pessimism of A compared to OO$^-$; the smaller the ratio, the less pessimistic the analysis. The results obtained with OO, OH and HH are presented in Table 1. For each application dimension, Table 1 presents the average ratio and the average execution times of the complete analysis (including all tasks and messages) in seconds. It is important to notice that, while the execution time is for the whole analysis, including all tasks and messages, the ratio is calculated only for the DYN messages, since their response time calculation is directly affected by the degree of pessimism of the various approaches proposed in the paper. The ratio calculated over all tasks and messages in the system is smaller than the ones shown in Table 1.

We can see that OO is very close to OO$^-$, which means that OO$^-$ is a good comparison baseline (it is only slightly optimistic). Due to the very large execution times, we were not able to run OO for more than 20 DYN messages.

Table 1 shows that OH produces very good quality results, in a reasonable time. For example, for 40 DYN messages, the analysis has finished in 367.87 seconds on average, and the average ratio is only 1.005.

Another result from Table 1 concerns the HH heuristic. Although HH is slightly more pessimistic than OH (for example, the DYN response times determined with HH were 1.012 times larger, on average, than those of OO$^-$ for applications with 30 messages, compared to 1.005 for OH), it is also significantly faster. We have successfully analyzed with HH large applications, with over 100 DYN messages in 0.16 seconds on average. Thus, HH is also suitable for design space exploration, where a potentially huge number of design alternatives have to be analyzed in a very short time.

As discussed in the Section 5.1.3, the quality of results obtained by the heuristic might influence the worst-case response times. In order to evaluate the pessimism introduced by not considering conditions (11)–(13) we have run a set of experiments with 15 applications of 40 tasks and 25 dynamic messages mapped on an architecture consisting of two nodes. Figure 6 presents the ratio for HH calculated according to Equation (19) as we vary the number of frame identifiers per processor from 2 to 6. We can see that the quality of the heuristic improves as the number of frame IDs increases (and, consequently, the number of messages sharing the same *FrameID* decreases). The more messages are sharing a *FrameID*, the more important conditions (11)–(13) are to the quality of the result, because they restrict the way bins can be covered (e.g., messages sharing the same *FrameID* should not be packed in the same bin). However, even for a small number of frame IDs HH produces good quality results (e.g., for two frame IDs, HH's ratio is 1.1226).

All of the experiments presented so far are on the calculation of response times for DYN messages. Our last set of experiments focused on the actual quality of $BusCycles_m$ heuristic from Section 5.1.3. We have considered 15 applications of 30 tasks with 15 DYN messages mapped on an architecture of three nodes. The ratio of $BusCycles_m^H$ from 5.1.3 over $BusCycles_m^O$ calculated as in 5.1.1 is 1.1014.

Finally, we considered a real-life example implementing a vehicle cruise controller that consists of 54 tasks mapped over 5 nodes, resulting in 26 DYN messages. We considered that 10 percent of the FlexRay communication cycle is allocated to the DYN segment communication. Scheduling the system using the OO approach took 0.19 seconds. Using the OH approach took 0.08 s, while the HH alternative was the fastest, finishing the analysis in 0.002 s. The average ratio of OH relative to OO is 1.003, while the average ratio of HH relative to OO is 1.004, which means that the heuristics obtained results almost identical to the optimal approach OO.

8. Conclusions

In this paper, we have presented a schedulability analysis for the FlexRay communication protocol. Timing properties of the ST messages have been established by building a static cyclic scheduling schedule, while for DYN messages we have, for the first time, developed a worst-case response time analysis. The FlexRay message analysis has been integrated in the context of a holistic schedulability analysis that determines the timing properties for all the tasks and messages in the system.

We have proposed three approaches for the derivation of worst-case response times of DYN messages. OO uses an ILP formulation to derive the optimal solution for the communication delay. HH uses heuristic-based upper-bounds for a bin-covering problem in order to quickly determine good quality response times. OH is able to further reduce the pessimism of HH by using an ILP formulation for part of the solution.

References

[1] G. Agrawal, B. Chen, W. Zhao, S. Davari, "Guaranteeing Synchronous Message Deadlines with the Token Medium Access Control Protocol", *IEEE Transactions on Computers*, 43(3), 327–339, 1994.

[2] S.F. Assman, D.S. Johnson, D.J. Kleitman, J.Y.-T. Leung, "On a Dual Version of the One-Dimensional Bin Packing Problem", *Journal of Algorithms*, 5, 502–525, 1984.

[3] J. Berwanger, M. Peller, R. Griessbach, *A New High Performance Data Bus System for Safety-Related Applications*, http://www.byteflight.de, 2000.

[4] R. Bosch GmbH, *CAN Specification Version 2.0*, 1991.

[5] G. Cena, A. Valenzano, "Performance analysis of Byteflight networks", *Proceedings of the IEEE International Workshop on Factory Communication Systems*, 157–166, 2004.

[6] E.G. Coffman Jr., R.L. Graham, "Optimal Scheduling for two Processor Systems", *Acta Informatica*, 1, 1972.

[7] S. Ding, N. Murakami, H. Tomiyama, H. Takada, "A GA-Based Scheduling Method for FlexRay Systems", *Proceedings of EMSOFT*, 2005.

[8] M. Labbe, G. Laporte, S. Martello, "An exact algorithm for the dual bin packing problem", *Operations Research Letters 17*, 9–18, 1995.

[9] Echelon, *LonWorks: The LonTalk Protocol Specification*, http://www.echelon.com

[10] H. Ermedahl, H. Hansson, M. Sjödin, "Response-Time Guarantees in ATM Networks", *Proceedings of the IEEE Real-Time Systems Symposium*, 274–284, 1997.

[11] FlexRay homepage: http://www.flexray-group.com, 2005.

[12] A. Hamann, R. Ernst, "TDMA Time Slot and Turn Optimization with Evolutionary Search Techniques", *Proceedings of the Design, Automation and Test in Europe Conference*, Volume 1, 312–317, 2005.

[13] K. Hoyme, K. Driscoll, "SAFEbus", *IEEE Aerospace and Electronic Systems Magazine*, 8(3), 34–39, 1992.

[14] International Organization for Standardization, "Road vehicles-Controller Area Network (CAN)—Part 4: Time-triggered communication", ISO/DIS 11898–4, 2002.

[15] H. Kirrmann, P. Zuber, "The IEC/EEE train communication network", *IEEE Micro*, 21(2), 81-92, 2001.

[16] H. Kopetz, G. Bauer , "The time-triggered architecture", *Proceedings of the IEEE*, 91(1), 112–126, 2003.

[17] *Local Interconnect Network Protocol Specification*, http://www.lin-subbus.org

[18] T. Meyerowitz, C. Pinello, A. Sangiovanni-Vincentelli, "A tool for describing and evaluating hierarchical real-time bus scheduling policies", *Proceedings of the Design Automation Conference*, 312–317, 2003.

[19] P. S. Miner, "Analysis of the SPIDER Fault-Tolerance Protocols", *Proceedings of the 5th NASA Langley Formal Methods Workshop*, 2000.

[20] N. Navet, Y. Song, F. Simont-Lion, C. Wilwert, "Trends in Automotive Communication Systems", *Proceedings of the IEEE*, 93(6), 1204–1223, 2005.

[21] J. C. Palencia, M. Gonzaléz Harbour, "Schedulability Analysis for Tasks with Static and Dynamic Offsets", *Proceedings of the Real-Time Systems Symposium*, 26–38, 1998.

[22] P. Pedreiras, L. Almeida, "Combining Event-Triggered and Time-Triggered Traffic in FTT-CAN: Analysis of the Asynchronous Messaging System", *Proceedings of the Workshop on Factory Communication Systems*, 67–75, 2000.

[23] P. Pop, P. Eles, Z. Peng, A. Doboli, "Scheduling with Bus Access Optimization for Distributed Embedded Systems", *IEEE Transactions on VLSI Systems*, 8(5), 472–491, 2000.

[24] P. Pop, P. Eles, Z. Peng, "Schedulability-Driven Communication Synthesis for Time-Triggered Embedded Systems", *Real-Time Systems Journal*, 24, 297–325, 2004

[25] P. Pop, P. Eles, Z. Peng, "Schedulability-Driven Frame Packing for Multi-Cluster Distributed Embedded Systems", *ACM Transactions on Embedded Computing Systems*, 4(1), 2005, 112–140.

[26] T. Pop, P. Eles, Z. Peng, "Schedulability Analysis for Distributed Heterogeneous Time Event-Triggered Real-Time Systems", *Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS 2003)*, 257–266, 2003.

[27] T. Pop, P. Pop, P. Eles, Z. Peng, "Optimization of Hierarchically Scheduled Heterogeneous Embedded Systems", *Proceedings of 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 67–71, 2005.

[28] Profibus International, *PROFIBUS DP Specification*, http://www.profibus.com

[29] J. Rushby, "Bus Architectures for Safety-Critical Embedded Systems", *Springer-Verlag Lecture Notes in Computer Science*, 2211, 306–323, 2001.

[30] SAE Vehicle Network for Multiplexing and Data Communications Standards Committee, *SAE J1850 Standard*, 1994.

[31] J. K. Strosnider, T. E. Marchok, "Responsive, Deterministic IEEE 802.5 Token Ring Scheduling", *Journal of Real-Time Systems*, 1(2), 133–158, 1989.

[32] K. Tindell, A. Burns, A. Wellings, "Calculating CAN Message Response Times", Control Engineering Practice, 3(8), 1163-1169, 1995.

[33] K. Tindell, J. Clark, "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems", Microprocessing & Microprogramming, 50(2–3), 1994.

[34] WorldFIP: Digital data communications for measurement and control - Fieldbus standard for use in industrial control systems. parts 1 to 6, IEC Standard 61158, 2003.