Schedulability-Driven Communication Synthesis for Time Triggered Embedded Systems

Paul Pop, Petru Eles and Zebo Peng
Linköping University, Sweden

Abstract

We present an approach to static priority preemptive process scheduling for the synthesis of hard real-time distributed embedded systems where communication plays an important role. The communication model is based on a time-triggered protocol. We have developed an analysis for the communication delays with four different message scheduling policies over a time-triggered communication channel. Optimization strategies for the synthesis of communication are developed, and the four approaches to message scheduling are compared using extensive experiments.

1 Introduction

Depending on the particular application, an embedded system has certain requirements on performance, cost, dependability, size, etc. For hard real-time applications the timing requirements are extremely important: failing to meet a deadline can potentially lead to a catastrophic event. Thus, in order to function correctly, an embedded system implementing such an application has to meet its deadlines. One of the typical application areas for such systems is that of safety-critical automotive applications, e.g. drive-by-wire, brake-by-wire [**?**].

Due to the complexity of embedded systems, hardware/software co-synthesis environments are developed to assist the designer in finding the most cost effective solution that, at the same time, meets the design requirements [**?**].

In this chapter we concentrate on the schedulability analysis and communication synthesis of embedded hard real-time systems which are implemented on distributed architectures. Process scheduling is based on a static priority preemptive approach while the bus communication is statically scheduled.

Preemptive scheduling of independent processes with static priorities running on single-processor architectures has its roots in the work of Liu and Layland [**?**]. The approach has been later extended to accommodate more general computational models and has also been applied to distributed systems [**?**]. The reader is referred to [**?**, **?**, **?**] for surveys on this topic. In [**?**] an earlier deadline first strategy is used for non-preemptive scheduling of processes with possible data dependencies. Preemptive and non-preemptive static scheduling are combined in the cosynthesis environment described in [**?**, **?**]. In many of the previous scheduling approaches researchers have assumed that processes are scheduled independently. However, this is not the case in reality, where process sets can exhibit both data and control dependencies. Moreover, knowledge about these dependencies can be used in order to improve the accuracy of schedulability analyses and the quality of produced schedules. One way of dealing with data dependencies between processes with static priority based scheduling has been indirectly addressed by the extensions proposed for the schedulability analysis of distributed systems through the use of the *release jitter* [**?**]. Release jitter is the worst case delay between the arrival of a process and its release (when it is placed in the run-queue for the processor) and can include the communication delay due to the transmission of a message on the communication channel.

In [?] and [?] time offset relationships and phases, respectively, are used in order to model data dependencies. Offset and phase are similar concepts that express the existence of a fixed interval in time between the arrivals of sets of processes. The authors show that by introducing such concepts into the computational model, the pessimism of the analysis is significantly reduced when bounding the time behaviour of the system. The concept of dynamic offsets has been later introduced in [?] and used to model data dependencies [?].

Currently, more and more real-time systems are used in physically distributed environments and have to be implemented on distributed architectures in order to meet reliability, functional, and performance constraints. However, researchers have often ignored or very much simplified aspects concerning the communication infrastructure. One typical approach is to consider communication processes as processes with a given execution time (depending on the amount of information exchanged) and to schedule them as any other process, without considering issues like communication protocol, bus arbitration, packaging of messages, clock synchronization, etc. These aspects are, however, essential in the context of safety-critical distributed real-time applications and one of our objectives is to develop a strategy which takes them into consideration for process scheduling. Many efforts dedicated to communication synthesis have concentrated on the synthesis support for the communication infrastructure but without considering hard real-time constraints and system level scheduling aspects [?, ?, ?]. Lower level communication synthesis aspects under timing constraints have been addressed in [?, ?]. We have to mention here some results obtained in extending real-time schedulability analysis so that network communication aspects can be handled. In [?], for example, the CAN protocol is investigated while the work reported in [?] considers systems based on the ATM protocol. Analysis for a simple time-division multiple access (TDMA) protocol is provided in [?] that integrates processor and communication schedulability and provides a "holistic" schedulability analysis in the context of distributed real-time systems. The problem of how to allocate priorities to a set of distributed tasks is discussed in [?]. Their priority assignment heuristic is based on the schedulability analysis from [?].

In this chapter we consider the time-triggered protocol (TTP) [?] as the communication infrastructure for a distributed real-time system. However, the research presented is also valid for any other TDMA-based bus protocol that schedules the messages statically based on a schedule table like, for example, the SAFEbus [?] protocol used in the avionics industry. [?] provides a comparison of bus architectures for safety-critical embedded systems, concluding that TTP "is unique in being used for both automobile applications, where volume manufacture leads to very low prices, and aircraft, where a mature tradition of design and certification for flight-critical electronics provides strong scrutiny of arguments for safety."

In this chapter, we consider that processes are scheduled according to a static priority preemptive policy. TTP has been classically associated with non-preemptive static scheduling of processes, mainly because of fault tolerance reasons [?]. In [?, ?] we have addressed the issue of non-preemptive static process scheduling and communication synthesis using TTP. However, considering preemptive priority based scheduling at the process level, with time triggered static scheduling at the communication level, can be the right solution under certain circumstances [?]. A communication protocol like TTP provides a global time base, improves fault-tolerance and predictability. At the same time, certain particularities of the application or of the underlying real-time operating system can impose a priority based scheduling policy at the process level.

The contribution of this chapter is threefold. First we develop a schedulability analysis for distributed processes with preemptive priority based scheduling considering a TTP-based communication infrastructure. As our second contribution, we have proposed four different approaches to message scheduling using static and dynamic message allocation. Finally, the third contribution is showing how the parameters of the communication protocol can be optimized in order to fit the communication particularities of a certain application. Thus, based on our approach, it is not only possible to determine if a certain task set implemented on a TTP-based distributed architecture is schedulable, but it is also possible to select a particular message passing strategy and also to optimize certain parameters of the communication protocol. By adapting the communication infrastructure to certain particularities of the task set we increase the likelihood of producing an implementation which satisfies all time constraints.

Figure 1: System Architecture

Figure 2: Bus Access Scheme

The chapter is divided into seven sections. The next section presents the architectures considered for system implementation. The computational model assumed and formulation of the problem are presented in Section **??**, and Section **??** presents the schedulability analysis for each of the four approaches considered for message scheduling. The optimization strategy is presented in Section **??**, and the four approaches are evaluated in Section **??**. The last section presents our conclusions.

2 System Architecture

2.1 Hardware Architecture

We consider architectures consisting of nodes connected by a broadcast communication channel. Every node consists of a TTP controller, a CPU, a RAM, a ROM and an I/O interface to sensors and actuators (Figure **??**). A node can also have an ASIC (Application Specific Integrated Circuit) in order to accelerate parts of its functionality.

Communication between nodes is based on the TTP [**?**]. TTP was designed for distributed real-time applications that require predictability and reliability (e.g, drive-by-wire). It integrates all the services necessary for fault-tolerant real-time systems.

The communication channel is a broadcast channel, so a message sent by a node is received by all the other nodes. The bus access scheme is time-division multiple-access (TDMA) (Figure **??**). Each node N_i can transmit only during a predetermined time interval, the so called TDMA slot S_i. In such a slot a node can send several messages packaged in a frame. A sequence of slots corresponding to all the nodes in the architecture is called a TDMA round. A node can have only one slot in a TDMA round. Several TDMA rounds can be combined together in a cycle that is repeated periodically. The sequence and length of the slots are the same for all TDMA rounds. However, the length and contents of the frames may differ.

Every node has a TTP controller that implements the protocol services and runs independently of the nodes CPU (Figure **??**). Communication with the CPU is performed through a so called message base interface (MBI) which is usually implemented as a dual ported RAM.

The TDMA access scheme is imposed by a so called message descriptor list (MEDL) that is located in every TTP controller. The MEDL basically contains: the time when a frame has to be sent or received, the address of the frame in the MBI and the length of the frame. MEDL serves as a schedule table for the TTP controller which has to know when to send or receive a frame to or from the communication channel.

The TTP controller provides each CPU with a timer interrupt based on a local clock synchronized with the local clocks of the other nodes. Clock synchronization is done by comparing the a priori known time of arrival of a frame with the observed arrival time. Thus, TTP provides a global time-base of known precision, without any overhead on the communication.

2.2 Software Architecture

We have designed a software architecture which runs on the CPU in each node and which has a real-time kernel as its main component. Each kernel has a so called tick scheduler that is activated periodically by the timer interrupts and decides on activation of processes, based on their priorities. Several activities, like polling of the I/O or diagnostics, also take place in the timer interrupt routine.

In order to run a predictable hard real-time application, the overhead of the kernel and the worst case administrative overhead (WCAO) of every system call have to be determined. Our

Figure 3: Message Passing Mechanism

schedulability analysis takes into account these overheads and also the overheads due to message passing.

The message passing mechanism is illustrated in Figure **??**, where we have three processes, P_1 to P_3. P_1 and P_2 are mapped on node N_0 that transmits in slot S_0, and P_3 is mapped on node N_1 that transmits in slot S_1. Message m_1 is transmitted between P_1 and P_2 that are on the same node, while message m_2 is transmitted from P_1 to P_3 between the two nodes.

Messages between processes located on the same processor are passed through shared protected objects. The overhead for their communication is accounted for by the blocking factor, using the analysis from [**?**] for the priority ceiling protocol.

Message m_2 has to be sent from node N_0 to node N_1. Thus, after m_2 is produced by P_1 it will be placed into an outgoing message queue, called *Out*. The access to the queue is guarded by a priority-ceiling semaphore. A so called transfer process (denoted by T in Figure **??**) moves the message from the *Out* queue into the MBI.

How the message queue is organized and how the message transfer process selects the particular messages and assembles them into a frame, depend on the particular approach chosen for message scheduling (see Section **??**). The message transfer process is activated, at certain a priori known moments, by the tick scheduler in order to perform the message transfer. These activation times are stored in a message handling time table (MHTT) available to the real-time kernel in each node. Both the MEDL and the MHTT are generated off-line as result of the schedulability analysis and optimization which will be discussed later. The MEDL imposes the times when the TTP controller of a certain node has to move frames from the MBI to the communication channel. The MHTT contains the times when messages have to be transferred by the message transfer process from the *Out* queue into the MBI, in order to be broadcasted by the TTP controller. As result of this synchronization, the activation times in the MHTT are directly related to those in the MEDL and the first table results directly form the second one.

It is easy to observe that we have the most favourable situation when, at a certain activation, the message transfer process finds in the *Out* queue all the "expected" messages which then can be packed into the immediate following frame to be sent by the TTP controller. However, application processes are not statically scheduled and availability of messages in the *Out* queue cannot be guaranteed at fixed times. Worst-case situations have to be considered, as will be shown in Section **??**.

Let us consider Figure **??** again. There we assumed a context in which the broadcasting of the frame containing message m_2 is done in the slot S_0 of *Round*2. The TTP controller of node N_1 knows from its MEDL that it has to read a frame from slot S_0 of *Round* 2 and to transfer it into its MBI. In order to synchronize with the TTP controller and to read the frame from the MBI, the tick scheduler on node N_1 will activate, based on its local MHTT, a so called delivery process, denoted with D in Figure **??**. The delivery process takes the frame from the MBI and extracts the messages from it. For the case when a message is split into several packets, sent over several TDMA rounds, we consider that a message has arrived at the destination node after all its constituent packets have arrived. When m_2 has arrived, the delivery process copies it to process P_3 which will be activated. Activation times for the delivery process are fixed in the MHTT just as explained earlier for the message transfer process.

The number of activations of the message transfer and delivery processes depends on the number of frames transferred, and it is taken into account in our analysis as also is the delay implied by the propagation on the communication bus.

3 Problem Formulation

We model an application as a set of processes. Each process P_i is allocated to a certain processor, and has a known worst case execution time C_i, a period T_i, a deadline D_i, and a uniquely assigned

priority. We consider a preemptive execution environment, which means that higher priority processes can interrupt the execution of lower priority processes. A lower priority process can block a higher priority process (e.g., it is in its critical section), and the blocking time is computed according to the priority ceiling protocol. Processes exchange messages, and for each message m_i we know its size S_{m_i}. A message is sent once in every n_m invocations of the sending process, with a period $T_m = n_m T_i$ inherited from the sender process P_i, and has a unique destination process. Each process is allocated to a node of the distributed system and messages are transmitted according to the TTP.

We are interested to synthesize the MEDL of the TTP controllers (and, as a direct consequence, also the MHTTs) so that the process set is schedulable on an as cheap (slow) as possible processor set.

The next section presents the schedulability analysis for each of the four approaches considered for message scheduling, under the assumptions outlined above. In Section **??**, the response times calculated using this schedulability analysis are combined in a cost function that measures the "degree of schedulability" of a given design alternative. This "degree of schedulability" is then used to drive the optimization and synthesis the MEDL and the MHTTs.

4 Schedulability Analysis

Under the assumptions presented in the previous section, [**?**] integrate processor and communication scheduling and provide a "holistic" schedulability analysis in the context of distributed real-time systems with communication based on a simple TDMA protocol. The validity of this analysis has been later confirmed in [**?**]. The analysis belongs to the class of response time analyses, where the schedulability test is whether the worst case response time of each process is smaller or equal than its deadline. In the case of a distributed system, this response time also depends on the communication delay due to messages. In [**?**] the analysis for messages is done is a similar way as for processes: a message is seen as an unpreemptable process that is "running" on a bus.

The basic idea in [**?**] is that the release jitter of a destination process depends on the communication delay between sending and receiving a message. The release jitter of a process is the worst case delay between the arrival of the process and its release (when it is placed in the run-queue for the processor). The communication delay is the worst case time spent between sending a message and the message arriving at the destination process.

Thus, for a process $d(m)$ that receives a message m from a sender process $s(m)$, the release jitter is

$$J_{d(m)} = r_{s(m)} + a_m + r_{deliver} + T_{tick} \tag{1}$$

where $r_{s(m)}$ is the response time of the process sending the message, a_m (worst case arrival time) is the worst case time needed for message m to arrive at the communication controller of the destination node, $r_{deliver}$ is the response time of the delivery process (see Section **??**), and T_{tick} is the jitter due to the operation of the tick scheduler. The communication delay for a message m (also referred to as the "response time" of message m) is

$$r_m = a_m + r_{deliver} \tag{2}$$

where a_m itself is the sum of the access delay Y_m and the propagation delay X_m. The access delay is the time a message queued at the sending processor spends waiting for the use of the communication channel. In a_m we also account for the execution time of the message transfer process (see Section **??**). The propagation delay is the time taken for the message to reach the destination processor once physically sent by the corresponding TTP controller. The analysis assumes that the period T_m of any message m is longer or equal to the length of a TDMA round, $T_m \geq T_{TDMA}$ (see Figures **??** and **??**).

The pessimism of this analysis can be reduced by using the notion of offset in order to model precedence relations between processes [**?**]. The basic idea is to exclude certain scenarios which are impossible due to precedence constraints. By considering dynamic offsets the tightness of the

analysis can be further improved [**?**, **?**]. Similar results have been reported in [**?**]. In [**?**] we have shown how such an analysis can be performed in the presence of not only precedence constraints but also control dependencies between processes, where the activation of processes depends on conditions computed by so-called "disjunction processes". In the present chapter our attention is concentrated on the analysis of network communication delays and on optimization of message passing strategies. In order to keep the discussion focused we present our analysis starting from the results in [**?**]. All the conclusions this research apply as well to the developments addressing precedence relations proposed, for example, in [**?**, **?**].

Although there are many similarities with the general TDMA protocol, the analysis in the case of TTP is different in several aspects and also differs to a large degree depending on the policy chosen for message scheduling.

Before going into details for each of the message scheduling approaches proposed by us, we analyze the propagation delay and the message transfer and delivery processes, as they do not depend on the particular message scheduling policy chosen. The propagation delay X_m of a message m sent as part of a slot S, with the TTP protocol, is equal to the time needed for the slot S to be transferred on the bus (this is the slot size expressed in time units, see Figure **??**). This time depends on the number of bits which can be packed into the slot and on the features of the underlying bus.

The overhead produced by the communication activities must be accounted for not only as part of the access delay for a message, but also through its influence on the response time of processes running on the same processor. We consider this influence during the schedulability analysis of processes on each processor. We assume that the worst case computation time of the transfer process (T in Figure **??**) is known, and that it is different for each of the four message scheduling approaches. Based on the respective MHTT, the transfer process is activated for each frame sent. Its worst case period is derived from the minimum time between successive frames.

The response time of the delivery process (D in Figure **??**), $r_{deliver}$, is part of the communication delay (Equation **??**). The influence due to the delivery process must be also included when analyzing the response time of the processes running on the respective processor. We consider the delivery process during the schedulability analysis in the same way as the message transfer process.

The response times of the communication and delivery processes are calculated, as for all other processes, using the arbitrary deadline analysis from [**?**].

The four approaches we propose for scheduling of messages using TTP differ in the way the messages are allocated to the communication channel (either statically or dynamically) and whether they are split or not into packets for transmission. The next subsections present the analysis for each approach as well as the degrees of liberty a designer has, in each of the cases, for optimizing the MEDL. First, we discuss each approach in the case when the arrival time a_m of a message m is smaller or equal with its period T_m. Then, in Section **??**, we present the generalization for the case $a_m > T_m$.

4.1 Static Single Message Allocation (SM)

The first approach to scheduling of messages using TTP is to statically (off-line) schedule each of the messages into a slot of the TDMA cycle, corresponding to the node sending the message. This means that for each message we decide off-line to allocate space in one or more frames, space that can only be used by that particular message. In Figure **??** the frames are denoted by rectangles. In this particular example, it has been decided to allocate space for message m in slot S_1 of the first and third rounds. Since the messages are dynamically produced by the processes, the exact moment a certain message is generated cannot be predicted. Thus, it can happen that certain frames will be left empty during execution. For example, if there is no message m in the *Out* queue (see Figure **??**) when the slot S_1 of the first round in Figure **??** starts, that frame will carry no information. A message m produced immediately after slot S_1 has left, could then be carried by the frame scheduled in the slot S_1 of the third round.

Figure 4: Worst case arrival time for SM

Figure 5: Optimizing the MEDL for SM and MM

In the SM approach, we consider that the slots can hold each maximum one single message. This approach is well suited for application areas, like safety-critical automotive electronics, where the messages are typically short and the ability to easily diagnose the system (fewer messages in a frame are easier to observe) is critical. In the automotive electronics area messages are typically a couple of bytes, encoding signals like vehicle speed. However, for applications using larger messages, the SM approach leads to overheads due to the inefficient utilization of slot space when transmitting smaller size messages.

As each slot carries only one fixed, predetermined message, there is no interference among messages. If a message m misses its allocated frame it has to wait for the following slot assigned to m. The worst case access delay Y_m for a message m in this approach is the maximum time between consecutive slots of the same node carrying the message m. We denote this time by θ_m, illustrated in Figure **??**, where we have a system cycle of length T_{cycle}, consisting of three TDMA rounds.

In this case, the worst case arrival time a_m of a message m becomes $\theta_m + X_m$. Therefore, the main aspect influencing schedulability of the messages is the way they are statically allocated to slots, which determines the values of θ_m. θ_m, as well as X_m, depend on the slot sizes which in the case of SM are determined by the size of the largest message sent from the corresponding node plus the bits for control and CRC, as imposed by the protocol.

As mentioned before, the analysis in [**?**], done for a simple TDMA protocol, assumes that $T_m \geq T_{TDMA}$. In the case of static message allocation with TTP (the SM and MM approaches), this translates to the condition $T_m \geq \theta_m$.

During the synthesis of the MEDL, the designer has to allocate the messages to slots in such a way that the process set is schedulable. Since the schedulability of the process set can be influenced by the synthesis of the MEDL only through the θ_m parameters, these are the parameters which have to be optimized.

Let us consider the simple example depicted in Figure **??**, where we have three processes, P_1, P_2, and P_3 running each on a different processor. When process P_1 finishes executing it sends message m_1 to process P_3 and message m_2 to process P_2. In the TDMA configurations presented in Figure **??**, only the slot corresponding to the CPU running P_1 is important for our discussion and the other slots are represented with light gray. With the configuration in Figure **??**a, where the message m_1 is allocated to the rounds 1 and 4 and the message m_2 is allocated to rounds 2 and 3, process P_2 misses its deadline because of the release jitter due to the message m_2 in *Round*2. However, if we have the TDMA configuration depicted in Figure **??**b, where m_1 is allocated to the rounds 2 and 4 and m_2 is allocated to the rounds 1 and 3, all the processes meet their deadlines.

4.2 Static Multiple Message Allocation (MM)

This second approach is an extension of the first one. In this approach we allow more than one message to be statically assigned to a slot and all the messages transmitted in the same slot are packaged together in a frame. As for the SM approach, there is no interference among messages, so the worst case access delay for a message m is the maximum time between consecutive slots of the same node carrying the message m, θ_m. It is also assumed that $T_m \geq \theta_m$.

However, this approach offers more freedom during the synthesis of the MEDL. We have now to decide also on how many and which messages should be put in a slot. This allows more flexibility in optimizing the θ_m parameter. To illustrate this, let us consider the same example depicted in Figure **??**. With the MM approach, the TDMA configuration can be arranged as depicted in Figure 5c, where the messages m_1 and m_2 are put together in the same slot in the rounds 1 and 2. Thus, the deadline is met and the release jitter is further reduced compared to the case presented

Figure 6: Optimizing the MEDL for DM and DP

in Figure **??**b where process P_3 was experiencing a large release jitter.

4.3 Dynamic Message Allocation (DM)

The previous two approaches have statically allocated one or more messages to their corresponding slots. This third approach considers that the messages are dynamically allocated to frames, as they are produced.

Thus, when a message is produced by a sender process it is placed in the *Out* queue (Figure **??**). Messages are ordered according to their priority. At its activation, the message transfer process takes a certain number of messages from the head of the *Out* queue and constructs the frame. The number of messages accepted is decided so that their total size does not exceed the length of the data field of the frame. This length is limited by the size of the slot corresponding to the respective processor. Since the messages are sent dynamically, we have to identify them in a certain way so that they are recognized when the frame arrives at the delivery process. We consider that each message has several identifier bits appended at the beginning of the message.

Since we dynamically package messages into frames in the order they are sorted in the queue, the access delay to the communication channel for a message m depends on the number of messages queued ahead of it.

The analysis in [**?**] bounds the number of queued ahead *packets* of messages of higher priority than message m, as in their case it is considered that a message can be split into packets before it is transmitted on the communication channel. We use the same analysis but we have to apply it for the number of *messages* instead of packets. We have to consider that messages can be of different sizes as opposed to packets which always are of the same size.

Therefore, the total *size* of higher priority messages queued ahead of a message m, in the worst case, is:

$$I_m = \sum_{\forall j \in hp(m)} \left\lceil \frac{r_{s(j)}}{T_j} \right\rceil S_j \tag{3}$$

where S_j is the size of the message m_j, $r_{s(j)}$ is the response time of the process sending message m_j, and T_j is the period of the message m_j.

Further, we calculate the worst case time that a message m spends in the *Out* queue. The number of TDMA rounds needed, in the worst case, for a message m placed in the queue to be removed from the queue for transmission is

$$\left\lceil \frac{S_m + I_m}{S_s} \right\rceil \tag{4}$$

where S_m is the size of the message m and S_s is the size of the slot transmitting m (we assume, in the case of DM, that for any message x, $S_x \leq S_S$). This means that the worst case time a message m spends in the *Out* queue is given by

$$Y_m = \left\lceil \frac{S_m + I_m}{S_s} \right\rceil T_{TDMA} \tag{5}$$

where T_{TDMA} is the time taken for a TDMA round.

Since the size of the messages is given with the application, the parameter that will be optimized during the synthesis of the MEDL is the slot size. To illustrate how the slot size influences schedulability, let us consider the example in Figure **??** where we have the same setting as for the example in Figure **??**. The difference is that we consider message m_1 having a higher priority than message m_2 and we schedule the messages dynamically as they are produced. With the configuration in Figure **??**a message m_1 will be dynamically scheduled first in the slot of the first round, while message m_2 will wait in the *Out* queue until the next round comes, thus causing

process P_2 to miss its deadline. However, if we enlarge the slot so that it can accommodate both messages, message m_2 does not have to wait in the queue and it is transmitted in the same slot as m_1. Therefore, P_2 will meet its deadline as presented in Figure **??**b. However, in general, increasing the length of slots does not necessarily improve schedulability, as it delays the communication of messages generated by other nodes.

4.4 Dynamic Packet Allocation (DP)

This approach is an extension of the previous one, as we allow the messages to be split into packets before they are transmitted on the communication channel. We consider that each slot has a size that accommodates a frame with the data field being a multiple of the packet size. This approach is well suited for the application areas that typically have large message sizes. By splitting messages into packets we can obtain a higher utilization of the bus and reduce the release jitter. However, since each packet has to be identified as belonging to a message, and messages have to be split at the sender and reconstructed at the destination, the overhead becomes higher than in the previous approaches.

The worst case time a message m spends in the *Out* queue is given by the analysis in [**?**] which is based on similar assumptions as those for this approach:

$$Y_m = \left\lceil \frac{p_m + I_m}{S_p} \right\rceil T_{TDMA} \tag{6}$$

where p_m is the number of packets of message m, S_p is the size of the slot (in number of packets) corresponding to m, and

$$I_m = \sum_{\forall j \in hp(m)} \left\lceil \frac{r_{s(j)}}{T_j} \right\rceil p_j \tag{7}$$

where p_j is the number of packets of a message m_j.

In the previous approach (DM) the optimization parameter for the synthesis of the MEDL was the size of the slots. With this approach we can also decide on the packet size which becomes another optimization parameter. Consider the example in Figure **??**c where messages m_1 and m_2 have a size of 6 bytes each. The packet size is considered to be 4 bytes and the slot corresponding to the messages has a size of 12 bytes (3 packets) in the TDMA configuration. Since message m_1 has a higher priority than m_2, it will be dynamically scheduled first in the slot of the first round and it will need 2 packets. In the third packet the first 4 bytes of m_2 are placed. Thus, the remaining 2 bytes of message m_2 have to wait for the next round, causing process P_2 to miss its deadline. However, if we change the packet size to 3 bytes and keep the same size of 12 bytes for the slot, we have 4 packets in the slot corresponding to the CPU running P_1 (Figure **??**d). Message m_1 will be dynamically scheduled first and will need 2 packets in the slot of the first round. Hence, m_2 can be sent in the same round so that P_2 can meet its deadline.

In this particular example, with one single sender processor and the particular message and slot sizes as given, the problem seems to be simple. This is, however, not the case in general. For example, the packet size which fits a particular node can be unsuitable in the context of the messages and slot size corresponding to another node. At the same time, reducing the packets size increases the overheads due to the transfer and delivery processes.

4.5 Arbitrary Arrival Times

The analysis presented in the previous sections is valid only in the case the arrival time a_m of a message m is smaller or equal with its period T_m. However, in the case $a_m > T_m$ the "arbitrary deadline" analysis from (Lehoczky 1990) has to be used. Lehoczky uses the notion of "level-i busy period" which is the maximum time for which a processor executes tasks of priority greater than or equal to the priority of process P_i. In order to find the worst case response time, a number of busy periods have to be examined.

The same analysis can be applied for messages. Thus, the worst case time message m takes to arrive at the communication controller of the destination node is determined in [**?**] using the arbitrary deadline analysis [**?**], and is given by:

$$a_m = \max_{q=0,1,2...} (w_m(q) + X_m(q) - qT_m) \tag{8}$$

where $Y_m(q) = w_m(q) - qT_m$ is the access delay, $X_m(q)$ is the propagation delay, and T_m is the period of the message. In the previous equation, q is the number of busy periods being examined, and $w_m(q)$ is the width of the level m busy period starting at time qT_m.

The approach to message scheduling in [**?**] is similar to the DP approach and considers that the messages are dynamically allocated to frames as they are produced, and that they can be split into packets before they are transmitted. Thus, the access delay (the time a message queued at the sending processor spends waiting for the use of the communication channel) is determined as being:

$$Y_m(q) = \left\lceil \frac{(q+1)p_m + I_m(w(q))}{S_p} \right\rceil T_{TDMA} - qT_m \tag{9}$$

with

$$I_m(w) = \sum_{\forall j \in hp(m)} \left\lceil \frac{w + r_{s(j)}}{T_j} \right\rceil p_j \tag{10}$$

where the variables have the same meaning as in the analysis presented in the previous section for the DP approach, and w is a function of m and q and denotes the width of the busy period.

Further, in order to determine the propagation delay $X_m(q)$ of a message m we have to observe that in the case of DP the messages can be split into packets, and thus the transmission of a message can span over several rounds. The analysis in [**?**] determines the propagation delay based on the number of packets that need to be taken from the queue over the time $w_m(q)$ in order to guarantee the transmission of the last packet of m. Thus, the propagation delay depends on the number of busy periods q being examined.

However, in the other three approaches, namely DM, MM and SM, messages cannot be split into packets and the transmission of a message is done in one single round. Therefore, the propagation delay X_m is equal to the slot size in which message m is being transmitted, and thus is not influenced by the number of busy periods being examined.

The access delay in the DM, MM and SM approaches is obtained through a particularization of the analysis for DP presented above. Thus, in the case of DM, in which messages are allocated dynamically but cannot be split into packets, we have:

$$Y_m(q) = \left\lceil \frac{(q+1)S_m + I_m(w(q))}{S_S} \right\rceil T_{TDMA} - qT_m \tag{11}$$

with

$$I_m(w) = \sum_{\forall j \in hp(m)} \left\lceil \frac{w + r_{s(j)}}{T_j} \right\rceil S_j \tag{12}$$

where the number of packets p_m and the size of the slot (measured in packets) S_p are replaced with the message size S_m and slot size S_s, respectively.

The analysis becomes even simpler in the case of the two static approaches. Since the allocation of messages to slots is statically performed in the SM and MM approaches there is no interference from other (higher priority) messages but only from later transmissions of the same message. Thus, the interference term I_m due to higher priority messages is null, and the access delay for both SM and MM is:

$$Y_m(q) = (q+1)\theta_m - qT_m. \tag{13}$$

Figure 7: Greedy Heuristic for SM

5 Optimization Strategy

Our problem is to analyze the schedulability of a given process set and to synthesize the MEDL of the TTP controllers (and consequently the MHTTs) so that the process set is schedulable on an as cheap as possible architecture. The optimization is performed on the parameters which have been identified for each of the four approaches to message scheduling discussed before. In order to guide the optimization process, we need a cost function that captures the "degree of schedulability" for a certain MEDL implementation. Our cost function is similar to that in [**?**] in the case an application is not schedulable (f_1). However, in order to distinguish between several schedulable applications, we have introduced the second expression, f_2, which measures, for a feasible design alternative, the total difference between the response times and the deadlines:

$$cost(optimization\ parameters) = \begin{cases} f_1 = \sum_{i=1}^{n} \max(0, R_i - D_i), \text{if } f_1 > 0 \\ f_2 = \sum_{i=1}^{n} R_i - D_i), \text{if } f_1 = 0 \end{cases} \quad (14)$$

where n is the number of processes in the application, R_i is the response time of a process P_i, and D_i is the deadline of a process P_i. If the process set is not schedulable, there exists at least one R_i that is greater than the deadline D_i, therefore the term f_1 of the function will be positive. In this case the cost function is equal to f_1. However, if the process set is schedulable, then all R_i are smaller than the corresponding deadlines D_i. In this case $f_1 = 0$ and we use f_2 as the cost function, as it is able to differentiate between two alternatives, both leading to a schedulable process set. For a given set of optimization parameters leading to a schedulable process set, a smaller f_2 means that we have improved the response times of the processes, so the application can be potentially implemented on a cheaper hardware architecture (with slower processors and/or bus, but without increasing the number of processors or buses).

The response time R_i is calculated according to the arbitrary deadline analysis [**?**] based on the release jitter of the process (see Section **??**), its worst case execution time, the blocking time, and the interference time due to higher priority processes. They form a set of mutually dependent equations which can be solved iteratively. As shown in [**?**], a solution can be found if the processor utilization is less than 100%.

For a given application, we are interested to synthesize a MEDL such that the cost function is minimized. We are also interested to evaluate in different contexts the four approaches to message scheduling, thus offering the designer a decision support for choosing the approach that best fits his application.

The MEDL synthesis problem belongs to the class of exponential complexity problems, therefore we are interested to develop heuristics that are able to find accurate results in a reasonable time. We have developed optimization algorithms corresponding to each of the four approaches to message scheduling. A first set of algorithms presented in Section **??** is based on simple and fast greedy heuristics. In Section **??** we introduce a second class of heuristics which aims at finding near-optimal solutions using the simulated annealing (SA) algorithm.

5.1 Greedy Heuristics

We have developed greedy heuristics for each of the four approaches to message scheduling. The main idea of the heuristics is to minimize the cost function by incrementally trying to reduce the communication delay of messages and, by this, the release jitter of the processes.

The only way to reduce the release jitter in the SM and MM approaches is through the optimization of the θ_m parameters. This is achieved by a proper placement of messages into slots (see Figure **??**).

Figure 8: Greedy Heuristic for DM

The `OptimizeSM` algorithm presented in Figure **??** starts by deciding on a size ($size_{S_i}$) for each of the slots. The slot sizes are set to the minimum size that can accommodate the largest message sent by the corresponding node (lines 1–4 in Figure **??**). In this approach a slot can carry at most one message, thus slot sizes larger than this size would lead to larger response times.

Then, the algorithm has to decide on the number of rounds, thus determining the size of the MEDL. Since the size of the MEDL is physically limited, there is a limit to the number of rounds (e.g., 2, 4, 8, 16 depending on the particular TTP controller implementation). However, there is a minimum number of rounds $MinRounds$ that is necessary for a certain application, which depends on the number of messages transmitted (lines 5–9). For example, if the processes mapped on node N_0 send in total 7 messages then we have to decide on at least 7 rounds in order to accommodate all of them (in the SM approach there is at most one message per slot). Several numbers of rounds, $RoundsNo$, are tried out by the algorithm starting from $MinRounds$ up to $MaxRounds$ (lines 15–31).

For a given number of rounds (that determine the size of the MEDL) the initially empty MEDL has to be populated with messages in such a way that the cost function is minimized. In order to apply the schedulability analysis that is the basis for the cost function, a *complete* MEDL has to be provided. A complete MEDL contains at least one instance of every message that has to be transmitted between the processes on different processors. A *minimal complete* MEDL is constructed from an empty MEDL by placing one instance of every message m_i into its corresponding empty slot of a round (lines 10–14). In Figure **??**a, for example, we have a MEDL composed of four rounds. We get a minimal complete MEDL, for example, by assigning m_2 and m_1 to the slots in rounds 3 and 4, and letting the slots in rounds 1 and 2 empty. However, such a MEDL might not lead to a schedulable system. The "degree of schedulability" can be improved by inserting instances of messages into the available places in the MEDL, thus minimizing the θ_m parameters. For example, in Figure **??**a by inserting another instance of the message m_1 in the first round and m_2 in the second round leads to P_2 missing its deadline, while in Figure **??**b inserting m_1 into the second round and m_2 into the first round leads to a schedulable system.

Our algorithm repeatedly adds a new instance of a message to the current MEDL in the hope that the cost function will be improved (lines 16–30). In order to decide an instance of which message should be added to the current MEDL, a simple heuristic is used. We identify the process P_i which has the most "critical" situation, meaning that the difference between its deadline and response time, $D_i - R_i$, is minimal compared with all other processes. The message to be added to the MEDL is the message $m = m_{P_i}$ received by the process P_i (lines 18–20). Message m will be placed into that round ($BestRound$) which corresponds to the smallest value of the cost function (lines 21–28). The algorithm stops if the cost function cannot be further improved by adding more messages to the MEDL.

The `OptimizeMM` algorithm is similar to `OptimizeSM`. The main difference is that in the MM approach several messages can be placed into a slot (which also decides its size), while in the SM approach there can be at most one message per slot. Also, in the case of MM, we have to take additional care that the slots do not exceed the maximum allowed size for a slot.

The situation is simpler for the dynamic approaches, namely DM and DP, since we only have to decide on the slot sizes and, in the case of DP, on the packet size. For these two approaches, the placement of messages is dynamic and has no influence on the cost function. The `OptimizeDM` algorithm (see Figure **??**) starts with the first slot $S_i = S_0$ of the TDMA round and tries to find that size ($BestSize_{S_i}$) which corresponds to the smallest $CostFunction$ (lines 4–14 in Figure **??**). This slot size has to be large enough ($S_i \geq MinSize_{S_i}$) to hold the largest message to be transmitted in this slot, and within bounds determined by the particular TTP controller implementation (e.g., from 2 bits up to $MaxSize = 32$ bytes). Once the size of the first slot has been determined, the algorithm continues in the same manner with the next slots (lines 7–12).

The `OptimizeDP` algorithm has also to determine the proper packet size. This is done by trying

Figure 9: The Simulated Annealing Strategy

all the possible packet sizes given the particular TTP controller. For example, it can start from 2 bits and increment with the "smallest data unit" (typically 2 bits) up to 32 bytes. In the case of the `OptimizeDP` algorithm the slot size has to be determined as a multiple of the packet size and within certain bounds depending on the TTP controller.

5.2 Simulated Annealing Strategy

We have also developed an optimization procedure based on a simulated annealing (SA) strategy. The main characteristic of such a strategy is that it tries to find the global optimum by randomly selecting a new solution from the neighbors of the current solution. The new solution is accepted if it is an improved one. However, a worse solution can also be accepted with a certain probability that depends on the deterioration of the cost function and on a control parameter called temperature [**?**].

In Figure **??** we give a short description of this algorithm. An essential component of the algorithm is the generation of a new solution x starting from the current one x^{now} (line 5 in Figure **??**). The neighbours of the current solution x^{now} are obtained depending on the chosen message scheduling approach. For SM, x is obtained from x^{now} by inserting or removing a message in one of its corresponding slots. In the case of MM, we have to take additional care that the slots do not exceed the maximum allowed size (which depends on the controller implementation), as we can allocate several messages to a slot. For these two static approaches we also decide on the number of rounds in a cycle (e.g., 2, 4, 8, 16; limited by the size of the memory implementing the MEDL). In the case of DM, the neighboring solution is obtained by increasing or decreasing the slot size within the bounds allowed by the particular TTP controller implementation, while in the DP approach we also increase or decrease the packet size.

For the implementation of this algorithm, the parameters TI (initial temperature), TL (temperature length), α (cooling ratio), and the stopping criterion have to be determined. They define the so called cooling schedule and have a strong impact on the quality of the solutions and the CPU time consumed. We were interested to obtain values for TI, TL and α that will guarantee the finding of good quality solutions in a short time. In order to tune the parameters we have first performed very long and expensive runs on selected large examples and the best ever solution, for each example, has been considered as the near-optimum. Based on further experiments we have determined the parameters of the SA algorithm, for different sizes of examples, so that the optimization time is reduced as much as possible but the near-optimal result is still produced. These parameters have then been used for the large scale experiments presented in the following section. For example, for the graphs with 320 nodes, TI is 300, TL is 500 and α is 0.95. The algorithm stops if for three consecutive temperatures no new solution has been accepted.

6 Experimental Results

For evaluation of our approaches we first used sets of processes generated for experimental purpose. We considered architectures consisting of 2, 4, 6, 8 and 10 nodes. 40 processes were assigned to each node, resulting in sets of 80, 160, 240, 320 and 400 processes. 30 process sets were generated for each of the five dimensions. Thus, a total of 150 sets of processes were used for experimental evaluation. Worst case computation times, periods, deadlines, and message lengths were assigned randomly within certain intervals. For the communication channel we considered a transmission speed of 256 kbps. The maximum length of the data field in a slot was 32 bytes and the frequency of the TTP controller was chosen to be 20 MHz. All experiments were run on a Sun Ultra 10 workstation.

For each of the 150 generated examples and each of the four message scheduling approaches we have obtained the near-optimal values for the cost function (Equation **??**) as produced by our SA

Figure 10: Comparison of the Four Approaches to Message Scheduling

Figure 11: Four Approaches to Message Scheduling: The Influence of the Message Sizes

based algorithm (see Section **??**). For a given example these values might differ from one message passing approach to another, as they depend on the optimization parameters and the schedulability analysis which are particular for each approach. Figure **??** presents a comparison based on the average percentage deviation of the cost function obtained for each of the four approaches, from the minimal value among them. The percentage deviation is calculated according to the formula:

$$deviation = \frac{cost_{approach} - cost_{best}}{cost_{best}} \times 100. \tag{15}$$

The DP approach is, generally, able to achieve the highest degree of schedulability, which in Figure **??** translates in the smallest deviation. In the case the packet size is properly selected, by scheduling messages dynamically we are able to efficiently use the available space in the slots, and thus reduce the release jitter. However, by using the MM approach we can obtain almost the same result if the messages are carefully allocated to slots as does our optimization strategy.

Moreover, in the case of larger process sets, the static approaches suffer significantly less overhead than the dynamic approaches. In the SM and MM approaches the messages are uniquely identified by their position in the MEDL. However, for the dynamic approaches we have to somehow identify the dynamically transmitted messages and packets. Thus, for the DM approach we consider that each message has several identifier bits appended at the beginning of the message, while for the DP approach the identification bits are appended to each packet. Not only the identifier bits add to the overhead, but in the DP approach, the transfer and delivery processes (see Figure **??**) have to be activated at each sending and receiving of a packet, and thus interfere with the other processes. Thus, for larger applications (e.g. process sets of 400 processes), MM outperforms DP, as DP suffers from large overhead due to its dynamic nature. DM performs worse than DP because it does not split the messages into packets, and this results in a mismatch between the size of the messages dynamically queued and the slot size, leading to unused slot space that increases the jitter. SM performs the worst as it does not permit much room for improvement, leading to large amounts of unused slot space. Also, DP has produced a MEDL that resulted in schedulable process sets for 1.33 times more cases than the MM and DM. MM, in its turn, produced two times more schedulable results than the SM approach.

Together with the four approaches to message scheduling a so called ad-hoc approach is presented. The ad-hoc approach performs scheduling of messages without trying to optimize the access to the communication channel. The ad-hoc solutions are based on the MM approach and consider a design with the TDMA configuration consisting of a simple, straightforward allocation of messages to slots. The lengths of the slots were selected to accommodate the largest message sent from the respective node. Figure **??** shows that the ad-hoc alternative is constantly outperformed by any of the optimized solutions. This demonstrates that significant gains can be obtained by optimization of the parameters defining the access to the communication channel.

Next, we have compared the four approaches with respect to the number of messages exchanged between different nodes and the maximum message size allowed. For the results depicted in Figures **??** and **??** we have assumed sets of 80 processes allocated to 4 nodes. Figure **??** shows that, as the number of messages increases, the difference between the approaches grows while the ranking among them remains the same. The same holds for the case when we increase the maximum allowed message size (Figure **??**), with a notable exception: for large message sizes MM becomes better than DP, since DP suffers from the overhead due to its dynamic nature.

Figure 12: Four Approaches to Message Scheduling: The Influence of the Message Sizes

		80 procs.	160 procs.	240 procs.	320 procs.	400 procs.
SM	avg.	0.12%	0.19%	0.50%	1.06%	1.63%
	max.	0.81%	2.28%	8.31%	31.05%	18.00%
MM	avg.	0.05%	0.04%	0.08%	0.23%	0.36%
	max.	0.23%	0.55%	1.03%	8.15%	6.63%
DM	avg.	0.02%	0.03%	0.05%	0.06%	0.07%
	max.	0.05%	0.22%	0.81%	1.67%	1.01%
DP	avg.	0.01%	0.01%	0.05%	0.04%	0.03%
	max.	0.05%	0.13%	0.61%	1.42%	0.54%

Table 1: Percentage Deviations for the Greedy Heuristics compared to SA

Figure 13: Hardware Architecture for the CC

We were also interested in the quality of our greedy heuristics. Thus, we have run all the examples presented above, using the greedy heuristics and compared the results with those produced by the SA based algorithm. Table **??** shows the average and maximum percentage deviations of the cost function values produced by the greedy heuristics from those generated with SA, for each of the graph dimensions. All the four greedy heuristics perform very well, with less than 2% loss in quality compared to the results produced by the SA algorithms. The execution times for the greedy heuristics were more than two orders of magnitude smaller than those with SA. Although the greedy heuristics can potentially find solutions not found by SA, for our experiments, the extensive runs performed with SA have led to a design space exploration that has included all the solutions produced by the greedy heuristics.

The above comparison between the four message scheduling alternatives is mainly based on the issue of schedulability. However, when choosing among the different policies, several other parameters can be of importance. Thus, a static allocation of messages can be beneficial from the point of view of testing and debugging and has the advantage of simplicity. Similar considerations can lead to the decision not to split messages. In any case, however, optimization of the bus access scheme is highly desirable.

6.1 Case Study

A typical safety critical application with hard real-time constraints, to be implemented on a TTP based architecture, is a vehicle cruise controller (CC). We have considered a CC system derived from a requirement specification provided by the industry. The CC described in this specification delivers the following functionality: it maintains a constant speed for speeds over 35 km/h and under 200 km/h, offers an interface (buttons) to increase or decrease the reference speed, and is able to resume its operation at the previous reference speed. The CC operation is suspended when the driver presses the brake pedal. The specification assumes that the CC will operate in an environment consisting of several nodes interconnected by a TTP channel (Figure **??**). There are five nodes which functionally interact with the CC system: the Anti Blocking System (ABS), the Transmission Control Module (TCM), the Engine Control Module (ECM), the Electronic Throttle Module (ETM), and the Central Electronic Module (CEM). It has been decided to distribute the functionality (processes) of the CC over these five nodes. The transmission speed of the communication channel is 256 kbps and the frequency of the TTP controller was chosen to be 20 MHz. We have modeled the specification of the CC system using a set of 32 processes and 17 messages. All the experiments were run on a Sun Ultra 10 workstation. For the given application the ad-hoc approach was unable to produce a schedulable solution. Then, we ran the simulated annealing strategy in order to determine if the CC system is schedulable with each of the four approaches to message scheduling. The SM approach failed to produce a schedulable solution after 304 seconds optimization time. However, with the other three approaches schedulable solutions could be produced. The smallest cost function resulted for the DP approach (taking 730 seconds

	Percentage Deviation from SA	Execution Time (sec)
SM	0.4%	23.08
MM	0.8%	87.98
DM	0.5%	21.37
DP	0.3%	104.75

Table 2: Results for the Cruise Controller Example

of optimization time), followed in this order by MM (1193 seconds optimization time) and DM (648 seconds optimization time).

Using the greedy strategies similar results have been obtained with significantly shorter optimization times. As with the SA-based algorithm, SM failed to produce a schedulable solution, while the other approaches succeded. Table **??** presents the percentage deviations of the cost function obtained with the greedy algorithms from the values obtained with SA, and the execution times.

7 Conclusions

In this chapter we have considered hard-real time systems implemented on distributed architectures consisting of several nodes interconnected by a communication channel. Processes are scheduled using a preemptive policy with fixed priorities. The bus access is statically scheduled according to the time triggered protocol. We have presented an approach to schedulability analysis with special emphasis on the impact of the communication infrastructure and protocol on the overall system performance.

We have considered four different approaches to message scheduling over TTP, that were compared based on the issue of schedulability. It has been also shown how the parameters of the communication protocol can be optimized in order to fit the communication particularities of a certain application. By optimizing the bus access scheme significant improvements can be obtained with regard to the "degree of schedulability" of the system. Experimental results show that our optimization heuristics are able to efficiently produce good quality results.