Real-Time Syst (2017) 53:916–956
DOI 10.1007/s11241-017-9283-6

Designing end-to-end resource reservations in predictable distributed embedded systems

Mohammad Ashjaei[1] · Nima Khalilzad[2] · Saad Mubeen[1] · Moris Behnam[1] · Ingo Sander[2] · Luis Almeida[3] · Thomas Nolte[1]

Published online: 15 June 2017

Abstract Contemporary distributed embedded systems in many domains have become highly complex due to ever-increasing demand on advanced computer controlled functionality. The resource reservation techniques can be effective in lowering the software complexity, ensuring predictability and allowing flexibility during the development and execution of these systems. This paper proposes a novel end-to-end resource reservation model for distributed embedded systems. In order to support the development of predictable systems using the proposed model, the paper provides a method to design resource reservations and an end-to-end timing analysis. The reservation design can be subjected to different optimization criteria with respect to runtime footprint, overhead or performance. The paper also presents and evaluates a case study to show the usability of the proposed model, reservation design method and end-to-end timing analysis.

Keywords End-to-end reservation · Response time analysis · Server design · Distributed embedded systems · Resource reservation

1 Introduction

Due to recent advances in computer-controlled functionality, the software in real-time systems has become overly complex. One way to deal with such complexity during the

✉ Mohammad Ashjaei
mohammad.ashjaei@mdh.se

1 Mälardalen University, Västerås, Sweden

2 KTH Royal Institute of Technology, Stockholm, Sweden

3 Instituto de Telecomunicaes, Universidade do Porto, Porto, Portugal

development is to split the system into several applications. The applications are then developed by different teams or tier-1 suppliers independently. However, the applications still share the system resources such as processors and network bandwidth. In order to support isolation and independence among the applications after their deployment, a fraction of the system resources is reserved for each application by means of so-called resource reservation (also called resource partitioning) techniques (Shin and Lee 2003; Deng and Liu 1997). For instance, a portion of processor time can be allocated to a specific application using the hierarchical reservation framework (Lipari and Bini 2003; Saewong et al. 2002. Similarly, there are several techniques to achieve temporal isolation among applications in the network domain. For instance, a multi-level hierarchical framework is presented in Santos et al. (2011) that supports resource reservations in switched Ethernet networks. As a result, the timing requirements of each application can be verified independently. The applications can be integrated together without a need for analyzing the complete system as long as the timing behavior of each part has been independently verified and the set of reservations is feasible.

Many industrial systems are in fact distributed real-time systems. This means that the applications in these systems span over more than one processor, while the processors communicate with each other via a field bus[1] or other kind of real-time network. The resource reservations for these applications include at least joint reservations on computation and communication resources. Such joint reservation is called end-to-end resource reservation. Curiously, not many recent works addressed end-to-end resource reservations. Two examples can be found in Lakshmanan and Rajkumar (2008), which proposes a distributed kernel to guarantee the end-to-end timeliness and Oliveira et al. (2014) that provides a distributed resource management system, called D-RES. Unlike the proposal presented in Lakshmanan and Rajkumar (2008), in our model the network resources are explicitly considered with their effects on the timing analysis. Moreover, we provide a new general model compared to the model presented in D-RES. Our model is more suitable for applications that are found in the automotive domain since they have complex transactions.

1.1 Paper contribution

The contributions in this paper[2] are listed as follows.

1. A new end-to-end resource reservation model that supports reservations on both computation and communication resources in distributed embedded systems. The model supports resource reservations on several real-time network protocols. The proposed model is the first comprehensive model to support end-to-end resource reservations in distributed transactions with various activation patterns such as trigger chains, data chains and mixed chains (see Sect. 3 for details). These activation patterns are commonly used in the industrial control systems (Feiertag et al. 2008; Mubeen et al. 2013).

[1] Typical name given to real-time networks developed specifically for use in industrial plants.

[2] This paper extends our previous work (Ashjaei et al. 2016).

2. An end-to-end timing analysis for distributed embedded systems that are developed using the proposed model. The analysis computes not only end-to-end response times, but also end-to-end data path delays in distributed transactions with reserved resources. In particular, the analysis computes the age and reaction delays which find their importance in the control systems and body electronics domains of the automotive industry.
3. A method to design reservations for both computation and communication resources. The method is based on the constraint satisfaction problem formulation which allows to design end-to-end reservations according to four different objectives, namely, minimum application footprint, best performance, minimum overhead and combination of footprint and performance.
4. We identify that the existing end-to-end path delay analysis (Feiertag et al. 2008) is not applicable to the networks that can autonomously initiate network transmissions, e.g., the HaRTES architecture (Santos et al. 2011). As an additional small contribution of this work, we propose a solution that can be used to apply the existing analysis to such protocols. This solution is equally applicable to the end-to-end delay analysis of the systems with or without resource reservations. The end-to-end resource reservation model is then adapted according to this solution.
5. A proof of concept is provided for the above contributions by conducting an automotive application case study. Using the case study, the resource reservation model, the reservation design method and the end-to-end timing analysis are evaluated. Moreover, within the case study, performance of the reservation design method is evaluated with respect to its sensitivity to the amount of time that it takes to find solutions under limited available resources.

1.2 Paper layout

The rest of the paper is organized as follows. The next section presents the related work. Section 3 describes preliminaries about the timing analysis. Section 4 presents the end-to-end resource reservation model. Section 5 describes the timing analysis. Section 6 presents the method to optimally design the reservations. Section 7 presents the case study and evaluation. Section 8 concludes the paper and discusses the future work.

2 Related work

In this section we discuss the previous works on resource reservation. Moreover, we specifically address the previous aims to design such a reservation with different targets in mind.

2.1 Resource reservation

There are many resource reservation techniques in the processor domain such as virtual machines (Barham et al. 2003) and reservation support for time-sensitive multime-

dia applications (Feng 2005). A common approach for resource reservations uses servers for scheduling tasks or messages. There are several works in the literature that propose various servers for real-time systems. These servers include hierarchical partitions (Saewong et al. 2002); deferrable servers (Strosnider et al. 1995); and sporadic servers (Sprunt et al. 1989). These works also provide response-time analysis for the mentioned servers respectively. An alternative tighter analysis for periodic servers is discussed in Lipari and Bini (2003). Whereas, the analysis for hierarchical partitioned scheduling is presented in Shin and Lee (2003). The later work considers a generic periodic server model for both fixed-priority and EDF scheduling algorithms. The analysis presented in Almeida and Pedreiras (2004) uses the same concept as in Shin and Lee (2003), however it covers more realistic task model with release jitter. Similarly, there are several resource reservation techniques, mostly relying on the servers, in the network domain such as in Santos et al. (2011).

Nevertheless, very few works have addressed the reservation of resources at the distributed system level, where the resources include both processor and network. Some of these works consider soft real-time systems targeting multimedia applications such as Sojka et al. (2011) and Dermler et al. (1996). An adaptive Quality of Service (QoS) control is developed in Cucinotta and Palopoli (2010) to control the reservation during run-time.

A distributed kernel framework is developed in Lakshmanan and Rajkumar (2008). It guarantees the satisfaction of end-to-end timing requirements in distributed real-time applications. The application model is based on a distributed task graph, where each application is divided into several sub-tasks that communicate with each other. Each sub-task is allocated to one processor and the end-to-end deadline is decomposed to local deadlines. The reservation on the network is achieved by traffic shaping. A general model, called Q-RAM (Rajkumar et al. 1997), provides sharing of resources among multiple applications by controlling QoS in the system. The Q-RAM model has been extended to cope with the systems with rapid changes (Ghosh et al. 2004). A distributed task model is presented in Lorente et al. (2006) where tasks are connected through Remote Procedure Calls (RPCs). In this model, the transactions can be time-triggered or event-triggered. A schedulability analysis is also presented to validate the predictability of the transactions. However, the main focus in these works is on distributed tasks while the network protocols have not received much attention. In our model, we explicitly consider network resources along with their impact on the end-to-end timing analysis.

A global resource management model, called D-RES (Oliveira et al. 2014), supports reservations on both nodes (processors) and networks with end-to-end timing guarantees. Although this model seemingly resembles the model presented in this work, there are significant dissimilarities between them. In comparison to Oliveira et al. (2014), in which applications follow a client-server paradigm with server reservations per client, an application in our model consists of multiple distributed transactions that contain tasks that invoke each other in a sequence. Another distinctive feature of our model is the support for various activation patterns in distributed transactions such as trigger, data and mixed chains. These chains have different end-to-end timing behaviors (see Sect. 3).

2.2 Reservation design

Some of the above mentioned proposals already address the problem of designing servers. For instance, the work in Almeida and Pedreiras (2004) discusses the problem of assigning server parameters with the goal of using least system resources. Search-based approaches are proposed for this purpose. The work in Lipari and Bini (2005) presents a technique to design periodic servers in a hierarchical framework targeting the schedulability of real-time applications. In order to find the best server parameters, the work defines a cost function according to the processor overhead. Within the hierarchical scheduling framework, the work in Davis and Burns (2008) investigates the problem of selecting the server parameters. Several techniques are presented in the work to select the server parameters given that the rest of the system parameters are known. The techniques include the binary search and greedy algorithms. Moreover, the recent work presented in Aminifar et al. (2016) addresses a server-based reservation mechanism for control applications. The goal for designing the servers in these applications is to provide optimum bandwidth such that the control tasks are stabilized. To solve the problem, the work proposes to use the Karush–Kuhn–Tucker (KKT) necessary conditions which are used in nonlinear programing.

In order to provide the server parameters, the approach in Aminifar et al. (2016) uses two steps. The first step provides the parameters of the leaf servers. Whereas, the second step generates the parameters for the intermediate and root servers. The reservation design solution presented in this paper is different from the above work in the following aspects. (I) We design the end-to-end reservation rather than one reservation. Since the timing behavior of applications are tightly coupled with all of application reservations, we cannot design each reservation individually. (II) We provide a generic constraint satisfaction model which allows designing reservations with respect to different performance metrics such as response time, age delay, reaction delay, resource usage and scheduling overhead.

The work in Fisher and Dewan (2009) presents a polynomial-time algorithm to allocate bandwidth to handle sporadic tasks that are scheduled by earliest deadline first with EDP server models. The algorithm gives an approximation on the partitions for real-time components. In the same context, the work presented in Wandeler and Thiele (2005) uses real-time calculus to design the servers based on a given interface. The algorithm computes demand and service curves for components in real-time systems.

3 End-to-end response-times and delays

In this section, first we discuss the end-to-end response times and delays in single-node systems. Then we move the discussion to distributed systems. Finally, we identify the limitations in the existing end-to-end delay analysis.

3.1 Single-node/uniprocessor embedded systems

When a system is modeled with chains of tasks then the schedulability of the system can be determined by calculating end-to-end response time and delays in each chain

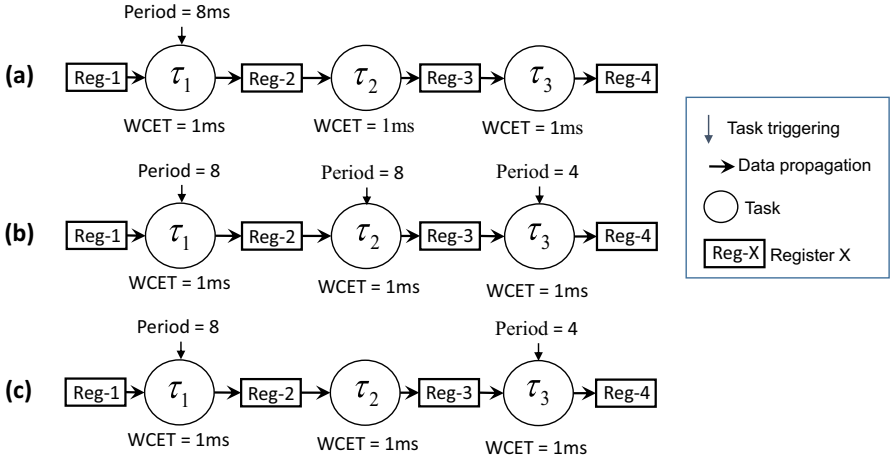


Fig. 1 Activation patterns: **a** a trigger chain, **b** a data chain, and **c** a mixed chain

and comparing them with the corresponding deadlines. In order to understand the difference between the end-to-end response time and end-to-end delays consider three task chains shown in Fig. 1. All task chains contain three tasks and each task is assumed to have the Worst Case Execution Time (WCET) of 1 ms. The tasks have equal priorities. The tasks communicate with each other by writing to and reading from the registers.[3]

There is only one activating source for the first chain, shown in Fig. 1a, that activates τ_1 periodically with a period of 8 ms. The rest of the tasks in the chain are activated by their predecessors. Such a chain is called a trigger chain (Mubeen et al. 2013). The input of the chain corresponds to the data read by τ_1 from register Reg-1. The input may correspond to the data that arrives from a sensor. On the other hand, the data written by τ_3 to Reg-4 is considered as the output of the task chain. The output may correspond to the control data for an actuator. The end-to-end response time of the task chain is defined as the amount of time elapsed between the arrival of an event at the first task and production of the response by the last task in the chain. The arrival of an event corresponds to the periodic activation of task τ_1. Assuming no interferences, the end-to-end response time of the chain can be calculated by summing up the WCETs of all tasks in the chain, i.e., 3 ms.

Now consider the task chain shown in Fig. 1b. In this chain, each and every task is activated independently, i.e, no task is activated by its predecessor. Such a chain is called a data chain (Mubeen et al. 2013). The periods of activation for tasks τ_1, τ_2 and τ_3 are 8, 8 and 4 ms respectively. Since each task in the chain is activated independently with a different clock, there can be several *time paths* through which the data can traverse from the input to the output. A time path is the path through which the data can propagate from the input to the output. Hence, there can be multiple outputs that correspond to one single input of the chain as shown by the uni-directional curved

[3] A register may correspond to a port of a software component which is realized by the task at run-time.

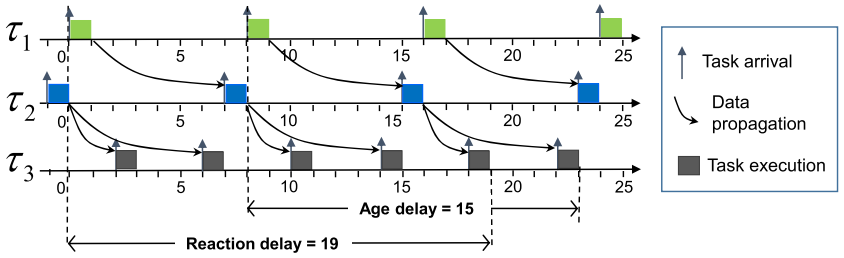


Fig. 2 Age and reaction delays in the scenario depicted in Fig. 1b

arrows in Fig. 2. As a result there are different delays between the arrival of the input data and production of the output data by the chain.

In this paper, we are interested in two such data delays, namely *age* and *reaction*, that have been identified in Feiertag et al. (2008). The age delay is equal to the time elapsed between the arrival of data at the input and the latest availability of the corresponding data at the output. For example, assume that the data is available in Reg-1 (input of the chain) at time 8 when it is read by the first task (τ_1). Task τ_1 writes the corresponding data to Reg-2 at time 9. The data is read by task τ_2 from Reg-2 at time 15. Task τ_2 then writes the corresponding data to Reg-3 at time 16. Task τ_3 reads the same data from Reg-3 at two different times, i.e., at 18 and 22. Moreover, task τ_3 writes the corresponding data to Reg-4 (output of the chain) at times 19 and 23. We can clearly see that there are two samples of output data corresponding to one single sample of the input data. In the age delay, we are interested in the longest time difference between the input data and the last sample of corresponding output data. On the other hand, the reaction delay corresponds to the earliest availability of the data at the first instance of the output corresponding to the data that *just missed* the read access at the input. Assume that just after task τ_1 started its execution at time 0, new data arrives in Reg-1. Hence, the new data just misses read access of τ_1. The new data has to wait in Reg-1 until it is read by τ_1 at time 8. The earliest effect of this data appears at the output at time 19 which corresponds to the reaction delay.

The age delay is important, in particular, for control applications where freshness of the data is of value. Whereas, the reaction delay is important in the applications where the first reaction to the input is of value, e.g., in the button-to-reaction applications in the body electronics domain. We refer the reader to Feiertag et al. (2008) and Mubeen et al. (2013) for the calculations of end-to-end response times and delays in single-node embedded systems without resource reservations.

In case of a trigger chain, e.g. shown in Fig. 1a, the age delay is equal to the end-to-end response time of the chain and the reaction delay is the summation of the age delay and one period of the first task in the chain. Figure 3 shows such a scenario.

It should be noted that a mixed chain includes both trigger and data sub-chains. An example of a mixed chain is depicted in Fig. 1c, where the first task is activated periodically with a period of 8 ms. Then, the second task is triggered by the first task, whereas the third task is activated periodically again with a period of 4 ms. The trace for this scenario is illustrated in Fig. 4. The age and reaction delays are also identified.

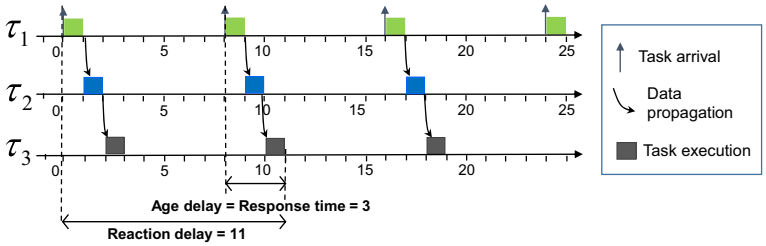


Fig. 3 Age and reaction delays in the scenario depicted in Fig. 1a

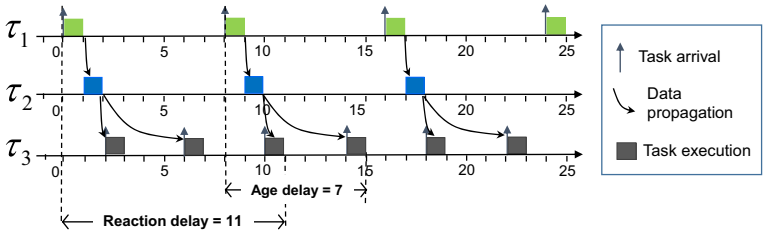


Fig. 4 Age and reaction delays in the scenario depicted in Fig. 1c

3.2 Distributed embedded systems

The delays, discussed in the previous subsection, are important not only in uniprocessor embedded systems but also in distributed embedded systems. Consider a data chain in a distributed system that consists of two nodes as shown in Fig. 5. There is only one task in each node. τ_1 in Node 1 is activated with a period of 6 ms. It sends m_1 which is received by τ_2 in Node 2. Similar to the example in Fig. 1, the tasks use registers for communication.

Assume that the network in Fig. 5 cannot initiate communication independent of the nodes. In other words, the network has no control over when a message can be queued for transmission at the network interface of a node. This is the case in majority of real-time network protocols such as Ethernet AVB (IEEE 2014), Flexray (Flexray Consortium 2010) and Controller Area Network (ISO 11898-1 1993). In this case, the message inherits the activation pattern and period from its sender task. Obviously, the message cannot be queued for transmission before the sending task has finished its execution. A possible execution trace of the example in Fig. 5 is depicted in Fig. 6a. The age and reaction delays are also identified. Once again, we refer the reader to Feiertag

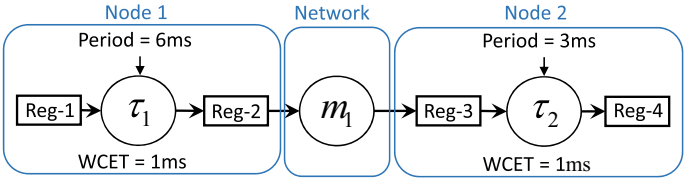


Fig. 5 A multi-rate chain in a distributed embedded system

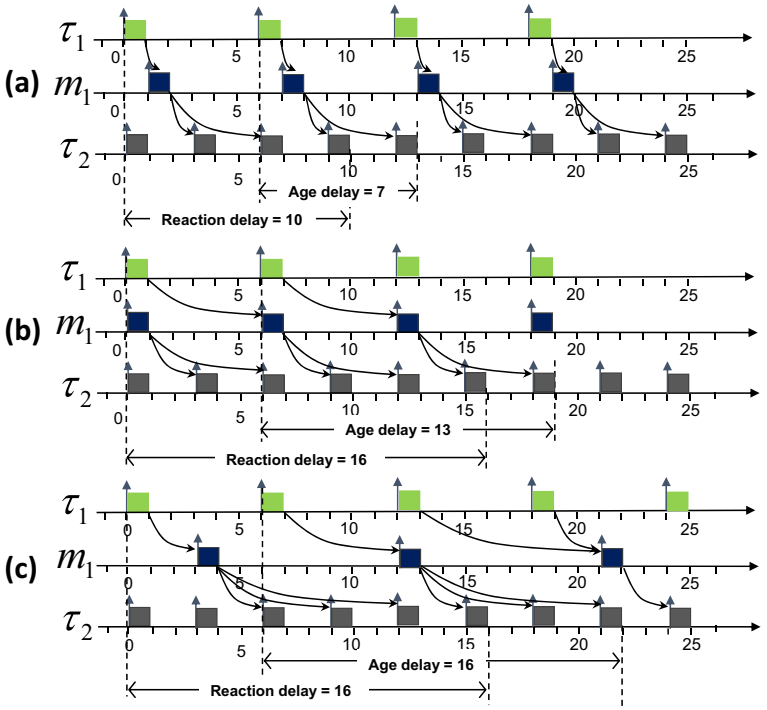


Fig. 6 Possible execution traces for the distributed embedded system example in Fig. 5

et al. (2008) and Mubeen et al. (2013) for the calculations of the age and reaction delays in distributed embedded systems without resource reservations.

3.3 Limitations in the existing analysis

Assume that the network in Fig. 5 allows transmission which is independent of the nodes, e.g., in the case of the Hard Real-Time Ethernet Switching (HaRTES) protocol Santos et al. (2011). This means that the network can transmit a message irrespective of whether the sender task has queued it in the network interface of the corresponding node or not. In the case, if the sender task has not queued the message then the older message is transmitted from Reg-2. Hence, it is possible that the sender task and the message are activated/queued for transmission at the same time, e.g., at time 0 in Fig. 6b. As a consequence, the age and reaction delays can be longer as it can be seen by comparing the corresponding delays in Fig. 6a and b. Note that all timing parameters in Fig. 6a and b are the same except for the dependent and independent activations of the message respectively. Even longer delays can be observed in the case when the period of independent activation of a message by the network is longer than the period of the sender task as shown in Fig. 6c. In this case, the maximum age delay (16 ms) is further increased by 3 ms compared to the case in Fig. 6b. In the case of a

multi-hop architecture, each link contributes such additional delay to the end-to-end delay.

The existing end-to-end delay analysis (Feiertag et al. 2008) abstracts the network protocols; hence it does not capture such independent activation behavior in the networks. In the timing analysis presented in this paper we consider the communication that is triggered independently of the tasks generating the messages. Section 5.3 provides a solution that lifts this limitation from the existing analysis.

4 End-to-end reservation model

We consider the model of a real-time distributed embedded system that consists of multiple nodes. The nodes are connected to a single network that allows real-time communication. The nodes are assumed to be single-core processors executing real-time tasks according to the fixed-priority preemptive scheduling. The system, denoted by $\mathcal{S}$, consists of N number of applications as shown in Eq. 1. An application performs a specific function and is commonly distributed over several nodes. For example, the cruise control function in a car is one application.

$$\mathcal{S} = \{\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(N)}\} \tag{1}$$

4.1 Application and transactional models

An application consists of a set of M transactions as shown in Eq. 2. A transaction is denoted by $\Gamma_i^{(h)}$, where i denotes its identifier and h denotes the identifier of the application to which it belongs. The transaction identifiers are unique within one application. However, two transactions belonging to different applications can have same transaction identifiers.

$$\mathcal{A}^{(h)} = \left\{\Gamma_1^{(h)}, \ldots, \Gamma_M^{(h)}\right\} \tag{2}$$

A transaction consists of a chain (or sequence) of real-time tasks and messages which can have different activation patterns, i.e., a task or a message can be activated (released for execution or transmission) independently (e.g., either periodically by a clock or sporadically by an external event) or by its predecessor task or message. The tasks and messages in transaction $\Gamma_i^{(h)}$ are presented in Eq. 3.

$$\Gamma_i^{(h)} = \{\tau_{j,i,k}, \ldots; m_{i,k}, \ldots\} \tag{3}$$

The subscript, k, in $\tau_{j,i,k}$ represents the task identifier, whereas the first and second subscripts, j and i, specify the identifiers of the node and transaction to which this task belongs. We allow any two transactions to share the same task or message, provided that the transactions belong to the same application. Note that sharing of tasks or messages between two applications is not allowed. A transaction can include more than one task from the same node. The task identifiers are assumed to be unique within each node.

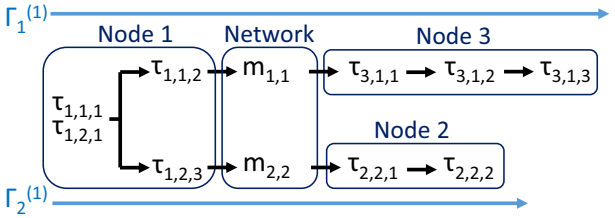


Fig. 7 An example of two transactions with task sharing

For instance, $\tau_{1,2,2}$ is task 2 in transaction 2 and it executes in node 1, whereas $\tau_{2,2,2}$ is also task 2 in transaction 2, however it executes in node 2. Moreover, in Eq. 3, $m_{i,k}$ represents message k in transaction i. Since we consider one network and assume the message identifiers to be unique, we do not use network identifier with the message.

It should be noted that a transaction cannot initiate or terminate with a message because a message must have a sender task and, at least, one receiver task. The transactional model is general in the sense that it can accommodate any number of tasks and messages that are in a chain. A transaction can also be composed of only one task, e.g., $\Gamma_1^{(1)} = \{\tau_{1,1,1}\}$. Such a transaction does not generate any message.

In order to understand the sharing of tasks among transactions, consider the example of a system that consists of three nodes and one network as shown in Fig. 7. There are three tasks in Node 1; two tasks in Node 2; three tasks in Node 3; and two messages in the network. There is one application in the system that consists of two transactions denoted by $\Gamma_1^{(1)}$ and $\Gamma_2^{(1)}$ as depicted in Fig. 7. Both of these transactions are initiated by the same task (i.e., the task with identifier 1 are shared between two transactions) in Node 1. This task is denoted by $\tau_{1,1,1}$ in transaction $\Gamma_1^{(1)}$ and by $\tau_{1,2,1}$ in transaction $\Gamma_2^{(1)}$. A shared task among two or more transactions can be identified by comparing the first and third subscripts of each task in one transaction with the first and third subscripts of each task in every other transaction within the same application. The sequence of tasks and messages included in $\Gamma_1^{(1)}$ and $\Gamma_2^{(1)}$ are as follows.

$\Gamma_1^{(1)}$: $\tau_{1,1,1} \rightarrow \tau_{1,1,2} \rightarrow m_{1,1} \rightarrow \tau_{3,1,1} \rightarrow \tau_{3,1,2} \rightarrow \tau_{3,1,3}$

$\Gamma_2^{(1)}$: $\tau_{1,2,1} \rightarrow \tau_{1,2,3} \rightarrow m_{2,2} \rightarrow \tau_{2,2,1} \rightarrow \tau_{2,2,2}$.

4.2 Task model

Each task is characterized by the following tuple.

$$\tau_{j,i,k} = \langle T_{j,i,k}, C_{j,i,k}, P_{j,i,k}, J_{j,i,k}, O_{j,i,k}, V_{j,i,k} \rangle \tag{4}$$

A task can be activated periodically or sporadically. In the case of periodic activation, $T_{j,i,k}$ represents its activation period. Whereas, in the case of sporadic activation, $T_{j,i,k}$ represents the minimum inter-arrival time between any two consecutive activations of the task. If the task is activated by its predecessor, a precedence constraint is specified between the task and its predecessor. Such information is stored in $V_{j,i,k}$,

which is null if the task is not activated by its predecessor. Note that in this model each task can have only one predecessor which activates the current task. The terms $C_{j,i,k}$, $P_{j,i,k}$ and $J_{j,i,k}$ in Eq. 4 represent the WCET, priority and maximum release jitter of the task. Note that the lower the value of $P_{j,i,k}$, the higher the priority. The offset for a periodic task is denoted by $O_{j,i,k}$. An offset is an externally imposed delay between the times when the task can be released for execution. Offsets for sporadic tasks are assumed to be zero. We assume that there is no synchronization among tasks, i.e., the tasks do not use locks to protect shared data.

4.3 Message model

A message is characterized by the following tuple.

$$m_{i,k} = \langle T_{i,k}, C_{i,k}, P_{i,k}, J_{i,k}, O_{i,k}, V_{i,k}, \mathcal{L}_{i,k} \rangle \tag{5}$$

A message can be queued for transmission either periodically or sporadically. If the message is periodic, $T_{i,k}$ represents its period. Whereas, if the message is sporadic then $T_{i,k}$ specifies the minimum inter-arrival time that should elapse between the queueing of its any two consecutive instances. The worst-case transmission time, priority and release jitter of the message are denoted by $C_{i,k}$, $P_{i,k}$ and $J_{i,k}$. Note that the higher priority is shown by the lower value for $P_{i,k}$. The offset of the message is denoted by $O_{i,k}$. The identifier of the task that sends the message $m_{i,k}$ is specified in $V_{i,k}$. The message model is applicable to several real-time network protocols such as real-time switched Ethernet extensions and Controller Area Network (CAN). In the case of switched Ethernet, we assume the communication is full duplex. This means that each network connection is composed of two independent unidirectional links with opposite directions. The message traverses through a set of links, denoted by $\mathcal{L}_{i,k}$, that are modeled in Eq. 6, where l_x and l_y are the first and last links the message crosses through. Note that the index of a link is unique within the network and they are shared among applications.

$$\mathcal{L}_{i,k} = \{l_x, \ldots, l_y\} \tag{6}$$

4.4 Resource model

Each application consumes a set of q resources. As shown in Eq. 7, $R^{(h)}$ denotes the resource set for hth application, whereas r_q represents the qth resource. The resource index, denoted by q, is unique within the system. Equation 7 includes node and communication resources. In a switched Ethernet network, one resource is defined per link.

$$R^{(h)} = \{r_1, \ldots, r_q\} \tag{7}$$

The total resources consumed by the system are obtained by performing the union operation on the resources consumed by all applications as shown in Eq. 8.

$$\mathcal{R} = \bigcup_{\forall h \in \mathcal{S}} \{R^{(h)}\} \tag{8}$$

The resources are allocated to the applications using the periodic resource reservation policy. According to this policy, each application periodically receives a portion of bandwidth in each resource that it requires. We define the interface of an application $\mathcal{A}^{(h)}$ as the properties of its reservations on q resources, and we denote the application interface using $I^{(h)}$:

$$I^{(h)} = \langle \{\Pi_1^{(h)}, \ldots, \Pi_q^{(h)}\}, \{\Theta_1^{(h)}, \ldots, \Theta_q^{(h)}\}, \{\Psi_1^{(h)}, \ldots, \Psi_q^{(h)}\} \rangle. \tag{9}$$

Above, the replenishment period for resource r_i is denoted by $\Pi_i^{(h)}$. The amount of budget to be replenished is denoted by $\Theta_i^{(h)}$. In addition, we define a priority for the resource reservation which is represented by $\Psi_i^{(h)}$ for resource r_i. The motivation behind the inclusion of the priority for each reservation is explained in the next subsection.

4.5 Node model

In order to support the resource reservation in nodes, we consider periodic servers (Strosnider et al. 1995). We assume that there is one server per application per node that schedules the tasks assigned to it. Each node implements a two-level hierarchical scheduler, where both levels use the fixed-priority preemptive scheduling. The top-level scheduler, called the *global level scheduler*, schedules the servers based on the priority of each reservation (Ψ). The second-level scheduler is called the *local level scheduler* that schedules the tasks assigned to it. Figure 8 shows an example of one node that implements a two-level hierarchical scheduler. The global level scheduler schedules two servers that schedule tasks belong to two applications within the node. Each server has its own local scheduler to schedule the tasks that are assigned to it.

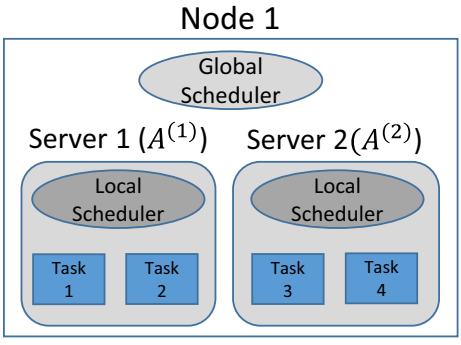


Fig. 8 Example of a node with a two-level hierarchical scheduler

4.6 Network model

For the network model any communication protocol that can support a resource reservation mechanism can be used. The response-time analysis for messages is specific for each network protocol. Therefore, as a proof of concept, we chose the HaRTES architecture (Ashjaei et al. 2014) because it allows independent activation of messages unlike the CAN and AVB networks. Moreover, the reservation mechanism in the HaRTES switch is more flexible than other technologies due to the hierarchical framework. Hence, we describe the HaRTES architecture in more details. The HaRTES switch is a modified Ethernet switch that provides real-time communication with a resource management mechanism. The HaRTES architecture is developed by connecting several HaRTES switches in a tree topology. The HaRTES architecture separates the traffic into two classes, synchronous (periodic) and asynchronous (sporadic). The transmission is performed within pre-configured Elementary Cycle (EC), which is a fixed-duration time slot. Each EC is divided into two transmission windows, one per traffic class. The EC along with the two windows for a link is shown in Fig. 9. Note that the size for the windows in each link can be different according to the load on the link.

The switch that is connected to the producer of a periodic message determines its transmission, making sure that it occurs within the associated window. The actual transmission is triggered by the switch with a Trigger Message (TM), transmitted within the Guard Window (see Fig. 9). The sporadic messages are transmitted whenever they are activated without waiting for the TM, yet within Asynchronous Window. The arriving message in the switch is buffered in a priority queue. The switch forwards the message as long as there is enough time in the associated window within the current EC. Otherwise, it is kept in the queue for the next EC. The HaRTES architecture allows consistent ECs in all links by time synchronizing all nodes and switches. For more details the reader is referred to Ashjaei et al. (2014). In this paper, we implement two servers per application per network resource (i.e., per link), one for Synchronous Window and the other for Asynchronous Window. It is important to dedicate a server per link for an application as each link in the network may share between several applications. Therefore, better tuning for the application resource allocation is possible. The reason behind using two servers per application per link is that the synchronous and asynchronous transmissions should be isolated to keep the fundamental feature of the HaRTES architecture. Therefore, if there are two messages in one application crossing a link and from different types, they are served by two different servers. If there is an application with one type of message (i.e., all messages are synchronous) only, then one server for the application per link is sufficient. Note that we may combine

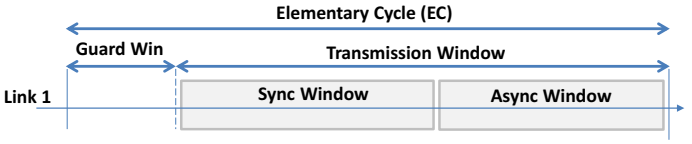


Fig. 9 The EC in the HaRTES architecture for two links

synchronous and asynchronous traffic that belong to the same application in one link to allocate one server for them. In such a case we loose the possibility of isolating the two types of traffic, while from the application perspective it is tolerated as both types belong to the same application.

4.7 Alternative network models

The end-to-end resource reservation model allows for alternative network technologies, provided these technologies support resource reservation mechanisms. In this section we discuss two network alternatives, Ethernet AVB (IEEE 2014) representing an Ethernet-based technology and CAN with server-based scheduling (Nolte et al. 2005) representing a broadcast communication scheme.

The Ethernet AVB standard foresees up to 8 classes of traffic. Commonly, two classes of traffic, i.e., classes A and B, are being used in automotive and automation domains, where class A has higher priority than class B. Within each class the transmission of messages is based on a single FIFO queue per output port. According to the standard, each class of traffic has a separate reservation per output port that is enforced by a traffic shaper called the Credit-Based Shaper (CBS). The CBS is allocated a part of total bandwidth to each class using *credits*. While the class has zero or positive credit, it can access the port for transmission, otherwise it is has to wait for replenishment of the credit. The increasing and decreasing rate of the credit is constant which is defined by a designer for each port according to the reserved bandwidth. In this technology, the link and message are defined according to the network model discussed in Sect. 4.6. However, in Ethernet AVB, two priority levels can be used only for classes A and B. The resource reservation model discussed in this paper can be applied to Ethernet AVB by assigning each application to separate class with a bandwidth reservation in each link. If two applications use the same class of traffic in one link, intuitively speaking, the applications loose their temporal isolation as they can interfere each other by using the same FIFO queue. Although the case of using Ethernet AVB introduces limitations on the number of applications per link, it can still be considered as an alternative technology for the proposed end-to-end reservation model.

In the server-scheduled CAN, there is one shared network resource corresponding to the CAN bus, which is in contrast to the Ethernet-based technologies with several network resources. According to the proposal presented in Nolte et al. (2005), multiple network servers (N-Servers) can be implemented on each node connected to the CAN bus. This concept resembles implementing multiple servers mediating access to a single resource (CAN bus). Each server is characterized by a period and it is allowed to send one message within the period. These servers are scheduled by a master server (M-Server) which is implemented in a particular node connected to the CAN bus. The M-Server is responsible for keeping track of the deadline that is defined for each N-Server. The M-Server is also responsible for allocating bandwidth for N-Servers. The M-Server further coordinates the access of N-Servers and provides isolations among them. We refer the reader to Nolte et al. (2005) for the details about the algorithms. The presented message model in this paper still holds for this network technology

with a difference that $\mathcal{L}_{i,k}$ contains no element. The resource model also holds where the resources are only nodes. There is no need to define a server with period, budget and priority for the network because the CAN bus is controlled through the nodes via N-Servers.

4.8 End-to-end timing requirements

The end-to-end timing requirements on each transaction $\Gamma_i^{(h)}$ belonging to application $\mathcal{A}^{(h)}$ are specified as a set of constraints, denoted by $Cr_i^{(h)}$. These constraints include an end-to-end deadline, denoted by $D_i^{(h)}$; an age constraint, denoted by $Age_i^{(h)}$; and a reaction constraint, denoted by $Reac_i^{(h)}$. It is up to the user to specify one or more of these constraints on each transaction.

$$Cr_i^{(h)} = \left\{ D_i^{(h)}, Age_i^{(h)}, Reac_i^{(h)} \right\} \tag{10}$$

4.9 System development model

We assume a system development model which involves the following two roles: (i) application developer; (ii) system integrator. Applications that are followed with the above model are developed by application developers. The application developers abstract the timing requirements of their application using the application interfaces (Eq. 9). The timing behavior of the applications, provided in an interface, is investigated using the analysis presented in Sect. 5. Designing interfaces, however, is addressed in Sect. 6. The system integrators, on the other hand, are responsible for integrating various applications while examining the timing correctness of the entire system. In this work we propose a compositional development model in the sense that the system integrators only need to use the application interfaces for investigating the timing properties of the system composed of several applications. The schedulability of the entire system can be done by a state of the art analysis given the interfaces by the application integrators.

4.10 Illustrative example

Consider an example of a distributed system with two applications, $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$. The system consists of two HaRTES switches that connect four nodes as shown in Fig. 10.

There are two transactions in $\mathcal{A}^{(1)}$ and one transaction in $\mathcal{A}^{(2)}$. The transactions for the two applications are as follows.

$\Gamma_1^{(1)} = \{\tau_{1,1,1}, \tau_{1,1,2}, m_{1,1}, \tau_{4,1,1}, \tau_{4,1,2}, \tau_{4,1,3}, m_{1,2}, \tau_{3,1,1}\}$

$\Gamma_2^{(1)} = \{\tau_{1,2,3}, \tau_{1,2,4}, m_{2,3}, \tau_{3,2,2}, \tau_{3,2,3}\}$

$\Gamma_1^{(2)} = \{\tau_{1,1,5}, m_{1,4}, \tau_{2,1,1}, \tau_{2,1,2}\}$

In Fig. 10, the node resources are identified by r_1 to r_4, whereas the communication resources are identified by r_5 to r_{10}. Note that the resources r_7 and r_8 represent two

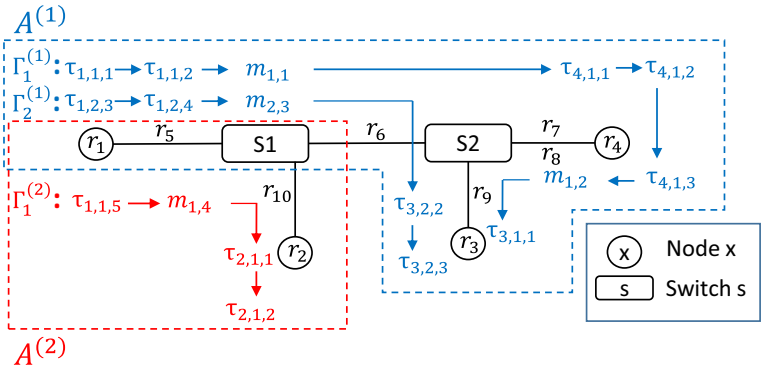


Fig. 10 An example to illustrate the system model

directions of the network connection (i.e., one for each link). The two applications share resources in Node 1, denoted by r_1, and the link between Node 1 and Switch 1, denoted by r_5. The resources used by the two applications are represented as follows.
$R^{(1)} = \{r_1, r_3, r_4, r_5, r_6, r_7, r_8, r_9\}$
$R^{(2)} = \{r_1, r_2, r_5, r_{10}\}$

According to the reservation model, a reservation should be defined for each resource that an application uses. For instance, the reservation in resource r_4 for application $\mathcal{A}^{(1)}$ is $\{\Pi_4^{(1)}, \Theta_4^{(1)}, \Psi_4^{(1)}\}$. This means that a server should be implemented in Node 4 with a period of $\Pi_4^{(1)}$, a budget of $\Theta_4^{(1)}$ and a priority of $\Psi_4^{(1)}$. This server schedules three tasks, namely $\tau_{4,1,1}$, $\tau_{4,1,2}$ and $\tau_{4,1,3}$ as shown in Fig. 10. Since r_1 is shared between the two applications, two servers should be implemented in Node 1. The first server, with parameters $\{\Pi_1^{(1)}, \Theta_1^{(1)}, \Psi_1^{(1)}\}$, is responsible for scheduling the four tasks in $\mathcal{A}^{(1)}$, namely $\tau_{1,1,1}$, $\tau_{1,1,2}$, $\tau_{1,2,3}$ and $\tau_{1,2,4}$, that belong to Node 1. Whilst, the second server, with parameters $\{\Pi_1^{(2)}, \Theta_1^{(2)}, \Psi_1^{(2)}\}$, schedules the only task, namely $\tau_{1,1,5}$, in $\mathcal{A}^{(2)}$ that belongs to Node 1.

5 Timing analysis

The system model (discussed in the previous section) strictly isolates the applications by resource reservations and by restricting sharing of tasks and messages among the applications. Therefore, schedulability of an application can be evaluated independently of the other applications. An application is said to be schedulable if all of its transactions are schedulable. That is, the end-to-end response times, age and reaction delays in each transaction satisfy their corresponding constraints. In this section, first we present the response-time analyses for tasks and messages conforming to the model discussed in the previous section. In these analyses, we consider the implicit deadline model, i.e., the task and message deadlines are assumed to be equal to the corresponding periods. Then we build upon these analyses and present the end-to-end response-time and delay analyses with resource reservations.

5.1 Response time analysis of tasks

We adapt the analysis presented in Almeida and Pedreiras (2004) as it is compatible with the presented model, e.g., the analysis is based on the assumption that the release jitter and deadline of each task are equal to or less than the task period. The analysis is based on a *supply bound function* (*sbf*) and a *request bound function* (*rbf*). The $sbf(t)$ is the minimum effective capacity that the resource provides within a time interval of $[0, t]$. On the other hand, the $rbf_i(t)$ is the maximum load generated by one task (τ_i) including the load of its higher priority tasks within the time interval of $[0, t]$. The earliest time where the $sbf(t)$ becomes equal to or larger than the $rbf_i(t)$ is the response time of τ_i. This method has been used in various models with different names for the supply and request bound functions.

The *sbf* for an application $\mathcal{A}^{(h)}$ in node j corresponding to the resource r_q is the lower bound on the availability of a server to handle the tasks belonging to the application. In order to sketch the lower bound, we must assume that the server is always available at the end of its period, except the first activation which is available at the beginning to make the worst case when the server is not available. Such a lower bound is depicted in Fig. 11. In the first period the server is available at the beginning of the period, while in the second period the server is available at the end of the period. Then, this patterns continues. Therefore, the *sbf* for each point in time has a value. As we assumed that the server is not available for almost two periods of the server, the supply bound function is zero.

Such a supply bound function, which is depicted in Fig. 11 is formulated in Eq. 11.

$$sbf_q^{(h)}(t) = \begin{cases} 0 & \text{if } t < \Delta \\ t - \left(\Delta + k \times \left(\Pi_q^{(h)} - \Theta_q^{(h)}\right)\right) & \text{if } \Delta + k \times \Pi_q^{(h)} \le t < \Delta + k \times \Pi_q^{(h)} + \Theta_q^{(h)} \\ (k+1) \times \Theta_q^{(h)} & \text{if } \Delta + k \times \Pi_q^{(h)} + \Theta_q^{(h)} \le t < \Delta + (k+1) \times \Pi_q^{(h)} \end{cases} \tag{11}$$

where $k = \lfloor (t - \Delta)/\Pi_q^{(h)} \rfloor$; while Δ is the worst-case latency of the server which is $2 \times (\Pi_q^{(h)} - \Theta_q^{(h)})$.

The *rbf* for the task $\tau_{j,i,k}$ that belongs to the application $\mathcal{A}^{(h)}$ in node j is presented in Eq. 12. The tasks that are shared between transactions are excluded by checking the condition $(k \neq v)$, i.e., if the task identifiers are not the same. If the interfering task $\tau_{j,p,v}$ is triggered by its predecessor, the period of its predecessor is extracted from

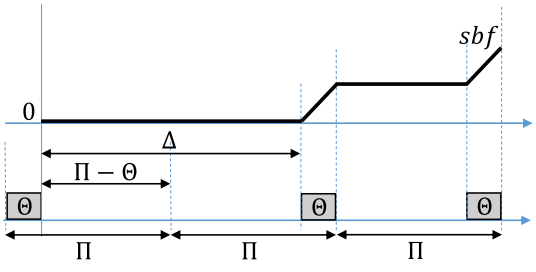


Fig. 11 The lower bound of the server availability

$V_{j,p,v}$. The extracted period is then used instead of $T_{j,p,v}$.

$$rbf^{(h)}_{j,i,k}(t) = C_{j,i,k} + \sum_{\substack{\forall \tau_{j,p,v} \in \mathcal{A}^{(h)} \\ \wedge\, k \neq v \wedge\, P_{j,p,v} \leq P_{j,i,k}}} \left\lceil \frac{t + J_{j,p,v}}{T_{j,p,v}} \right\rceil C_{j,p,v} \tag{12}$$

The response time of the task $\tau_{j,i,k}$, denoted by $RT_{j,i,k}$, is derived from Eq. 13. The inequality should be evaluated for all t in the interval that is equal to the period of $\tau_{j,i,k}$.

$$RT_{j,i,k} = min(t > 0) : sbf^{(h)}_{q}(t) \geq rbf^{(h)}_{j,i,k}(t) \tag{13}$$

5.2 Response time analysis of messages

Among the existing analyses, the closest analysis that could be applied to our model is the response-time analysis for a single-switch architecture (Santos et al. 2011), which is based on Explicit Deadline Periodic (EDP) resource model (Easwaran et al. 2007). However, the presented model considers that the messages traverse through multiple HaRTES switches with resource reservations. We present analysis for the multi-hop HaRTES architecture by adapting the existing analysis and using the *sbf* for the servers presented in Shin and Lee (2003). A server is implemented for each network resource to schedule the messages that belong to one application, provided that the server is used during either Synchronous or Asynchronous Windows which depends upon the activation patterns of the messages. The scheduling in the network link is performed in a hierarchical fashion, where the EC and its two windows constitute the top level while the servers within the windows represent the second level in the hierarchy. We provide the *sbf* for a message $m_{i,k}$ belonging to transaction i in application $\mathcal{A}^{(h)}$ that crosses link l_q, which is denoted by $sbf^{(h)}_{i,k,q}(t)$. Note that in the analysis we consider that messages are not larger than the maximum Ethernet size, hence there is no fragmentation of messages.

Since the messages cannot be preempted during their transmission, a portion of the budget in the server can be wasted. We define this wasted time as the *idle time*. Figure 12 shows a scenario in which a message cannot fit within the remaining budget in the first period of the server, hence it has to be sent after the replenishment during the next period. The idle time is upper bounded by the maximum size of the message in the set that includes $m_{i,k}$ and all the higher priority messages that cross the link l_q; belong to the same application; and have the same activation pattern as that of $m_{i,k}$. The idle time is denoted by $Id^{(h)}_{i,k,q}$ and is calculated in Eq. 14, where $AP(m_{i,k})$ presents the activation pattern for $m_{i,k}$.

$$Id^{(h)}_{i,k,q} = \max_{\substack{\forall m_{z,p} \in \mathcal{A}^{(h)} \wedge\, P_{z,p} \leq P_{i,k} \wedge\, l_q \in \mathcal{L}_{z,p} \\ \wedge\, p \neq k \wedge\, AP(m_{z,p}) = AP(m_{i,k})}} \{C_{z,p}\} \tag{14}$$

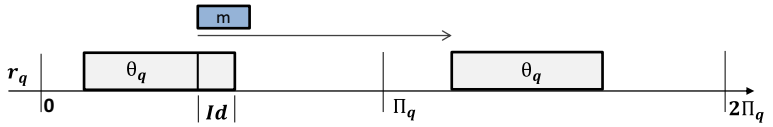


Fig. 12 Example demonstrating the inserted idle time

Therefore, in the calculation of the *sbf*, the idle time should be subtracted from the budget, i.e., $\Theta_q^{\prime(h)} = \Theta_q^{(h)} - Id_{i,k,q}^{(h)}$. The *sbf* can be computed as follows.

$$sbf_{i,k,q}^{(h)}(t) = \begin{cases} y.\Theta_q^{\prime(h)} + max\{t - x - y \cdot \Pi_q^{(h)}, 0\} & \text{if } t \geq \Pi_q^{(h)} - \Theta_q^{\prime(h)} \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

where $x = 2(\Pi_q^{(h)} - \Theta_q^{\prime(h)})$ and $y = \left\lfloor \frac{t-(\Pi_q^{(h)}-\theta_q^{\prime(h)})}{\Pi_q^{(h)}} \right\rfloor$.

The *rbf* is defined as the maximum load generated by a message with respect to its critical instant. The critical instant, in this case, corresponds to releasing the message with all its higher priority messages at the same time. The interference and blocking received by a message in the HaRTES architecture has been derived in Ashjaei et al. (2014). These interferences for $m_{i,k}$ are categorized as follows: (i) the interference from the higher and equal priority messages that share the link with the message, denoted by $I_{i,k,q}^{(h)}$, and (ii) the blocking from the lower priority messages that share the link with the message, denoted by $B_{i,k,q}^{(h)}$. The *rbf* for the message $m_{i,k}$ belonging to the application $\mathcal{A}^{(h)}$ crossing the link l_q is presented in Eq. 16.

$$rbf_{i,k,q}^{(h)}(t) = C_{i,k} + I_{i,k,q}^{(h)} + B_{i,k,q}^{(h)} \tag{16}$$

The interference of higher or equal priority messages is calculated according to Eq. 17. Similar to the case of response-time analysis for tasks, the messages that are shared among transactions should be excluded by checking the message identifier, i.e., $p \neq k$.

$$I_{i,k,q}^{(h)} = \sum_{\substack{\forall m_{z,p} \in \mathcal{A}^{(h)} \wedge\ P_{z,p} \leq P_{i,k} \wedge\ p \neq k \\ \wedge\ l_q \in \mathcal{L}_{z,p} \wedge\ AP(m_{z,p}) = AP(m_{i,k})}} \left\lceil \frac{t}{T_{z,p}} \right\rceil C_{z,p} \tag{17}$$

When a message is ready to be forwarded in the switch, it might be blocked by a lower priority message that is already under transmission. The blocking delay is the maximum size of the message within the messages that belong to the same application as $m_{i,k}$ and cross the link l_q.

$$B_{i,k,q}^{(h)} = \max_{\substack{\forall m_{z,p} \in \mathcal{A}^{(h)} \wedge\ P_{z,p} > P_{i,k} \wedge\ p \neq k \\ \wedge\ l_q \in \mathcal{L}_{z,p} \wedge\ AP(m_{z,p}) = AP(m_{i,k})}} \{C_{z,p}\} \tag{18}$$

The response time of $m_{i,k}$ crossing the link l_q is computed in Eq. 19. The period of the message should be decomposed for each link. This can be done in proportion to the load of the links or by equally distributing the load over the links. We refer the reader to the existing works (e.g., Chatterjee and Strosnider 1995) for details about the decomposition of deadlines. Equation 19 should be evaluated for all t until the decomposed deadline for link l_q.

$$RT_{i,k,q} = min(t > 0) : sbf^{(h)}_{i,k,q}(t) \geq rbf^{(h)}_{i,k,q}(t) \tag{19}$$

The response time of $m_{i,k}$ is derived in Eq. 20, where ϵ represents the switch fabric delay.

$$RT_{i,k} = \sum_{q=1}^{|\mathcal{L}_{i,k}|} RT_{i,k,q} + |\mathcal{L}_{i,k}| \times \epsilon \tag{20}$$

5.3 Timing analysis of transactions

The end-to-end response time for a transaction is the sum of the response times of tasks and messages that belong to it. The transaction is schedulable if its end-to-end response time, denoted by $RT^{(h)}_i$, is less than or equal to its end-to-end deadline, denoted by $D^{(h)}_i$. The end-to-end response time for transaction $\Gamma^{(h)}_i$ is computed as follows.

$$RT^{(h)}_i = \sum_{\forall \tau_{j,i,k} \in \Gamma^{(h)}_i} RT_{j,i,k} + \sum_{\forall m_{i,k} \in \Gamma^{(h)}_i} RT_{i,k} \tag{21}$$

When the tasks and messages in a transaction have independent activation sources, the age and reaction delays should be computed. In order to calculate these delays, all *reachable* time paths for a transaction should be derived. For example, in Fig. 2 one reachable time path is τ_1 (first instance), τ_2 (second instance) and τ_3 (third instance). Note that the time paths that are not reachable are excluded from the analysis, e.g., τ_1 (second instance), τ_2 (second instance) and τ_3 (third instance). A set of Boolean functions are presented in Feiertag et al. (2008) to derive such reachable time paths. However, time paths presented in Feiertag et al. (2008) are only applicable for networks in which the messages cannot be independently initiated (see Sect. 3.3). In the HaRTES architecture a message may be triggered independent of the sender task in case the message is synchronous message in the HaRTES. Therefore, independent activations of the messages in the transaction should be included in the time paths. Figure 13 shows a transaction with two tasks and a message. The message is triggered by the switch regardless of the sender task (τ_1). In this figure, five time paths (identified by A to E) are illustrated. However, not all of these time paths are reachable. For example, time path D is not reachable as the data cannot travel back in time.

The algorithm to compute the delays for $\Gamma^{(h)}_i$ is presented in Algorithm 1. In order to compute the longest time path, all reachable time paths are extracted based on the activation patterns of the tasks and messages in the transaction. The extraction is

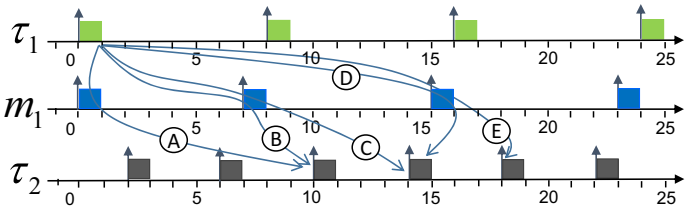


Fig. 13 An example of time paths including a message

done by the two functions in lines 1 and 2. Basically, every valid time path is verified against the reachability condition. This condition for two neighboring tasks (or a task and a message) is as follows. Let $\alpha_w(i)$ and $\alpha_r(i)$ denote the activation times of the ith instances of the reader and writer tasks respectively. Also, let $RT_w(i)$ denotes the response time of the writer task.

1. The activation time of the writer should be earlier than the reader, i.e., $\alpha_r(i) \geq \alpha_w(i)$. For instance, time path D in Fig. 13 does not satisfy this condition between m_1 and τ_2.
2. The execution of the writer and reader should not overlap, i.e., $\alpha_r(i) \geq \alpha_w(i) + RT_w(i)$. For example, time path A in Fig. 13 does not satisfy this condition between τ_1 and m_1.
3. The writer and reader can have overlap only if both of them execute on the same node or link, and the priority of the reader is lower than the priority of the writer.
4. There could be a time path in which the output of an instance of the writer is over-written by its next instance. For example, time path E in which the second instance of τ_1 over-writes the data from its previous instance. Such time paths are excluded from the list of reachable time paths.

In order to compute the reaction delay, a subset of reachable time paths is extracted. In this subset, no time path exists that shares the same start instance of the first task in the transaction and has an earlier end instance of the last task in the transaction.

The function in line 5 of the algorithm provides the age delay of the transaction for a given time path. The function uses Eq. 22 to compute the age delay (Feiertag et al. 2008), where α_n and α_1 represent the activation times of the last task and the first task in the transaction. Note that a transaction cannot start or finish with a message.

$$New\_\_Age = \alpha_n(TP) + RT_{j,i,k} - \alpha_1(TP) \tag{22}$$

The reaction delay derived in line 11 of the algorithm for a given time path is computed using Eq. 23, where $Pred(TP)$ is the instance of the first task in the transaction that belongs to the previous reachable time path.

$$New\_\_Reac = \alpha_n(TP) + RT_{j,i,k} - \alpha_1(Pred(TP)) \tag{23}$$

Algorithm 1 Find the age and reaction delays for $\Gamma_i^{(h)}$

```
TP__Age = FindAllValidAgeTimePaths(i)
TP__Reac = FindAllValidReacTimePaths(i)
Age__Delay_i^(h) = 0; Reac__Delay_i^(h) = 0
for all TP__Age do
    New__Age = ComputeAge(TP__Age)
    if New__Age > Age__Delay_i^(h) then
        Age__Delay_i^(h) = New__Age
    end if
end for
for all TP__Reac do
    New__Reac = ComputeReac(TP__Reac)
    if New__Reac > Reac__Delay_i^(h) then
        Reac__Delay_i^(h) = New__Reac
    end if
end for
return Age__Delay_i^(h), Reac__Delay_i^(h)
```

In Eqs. 22 and 23, $RT_{j,i,k}$ represents the response time of the last task in $\Gamma_i^{(h)}$ which executes in node j. The transaction meets its constraints if:

$$Age\_\_Delay_i^{(h)} \leq Age_i^{(h)} \;\&\; Reac\_\_Delay_i^{(h)} \leq Reac_i^{(h)} \tag{24}$$

5.4 Schedulability of applications

In order to verify the schedulability of applications, the response time of servers corresponding to each application should be evaluated on both nodes and network links. The servers are schedulable if their response times are less than or equal to their respective periods. The response time of a server for $\mathcal{A}^{(h)}$ in a node corresponds to resource r_q. It is recursively calculated using Eq. 25.

$$RT_q^{(h)} = \sum_{\forall \mathcal{A}^{(f)} \in \mathcal{S} \wedge\ \Psi_q^{(f)} \leq \Psi_q^{(h)}} \left\lceil \frac{RT_q^{(h)}}{\Pi_q^{(f)}} \right\rceil \Theta_q^{(f)} \tag{25}$$

On the network links, the servers use a portion of bandwidth, i.e., in the Synchronous or Asynchronous Windows. Therefore, the analysis based on the *sbf* and *rbf* is used. The Synchronous Window becomes available every EC at a known point in time, which is always after the Guard Window. Figure14 shows the availability of windows and the *sbf* for the Synchronous Window in link l_q, where L_{EC} is the size of the EC; and LS_q and LA_q are the sizes of the Synchronous and Asynchronous Windows in link l_q respectively. In order to calculate the supply for a given time interval t, we consider that the interval starts at the beginning of the EC as the resource is periodic in a fixed position within the EC. There are three scenarios for the time interval: (a) it finishes before the window, (b) it finishes within the window, and (c) it finishes before the end of the EC but consumes the whole window.

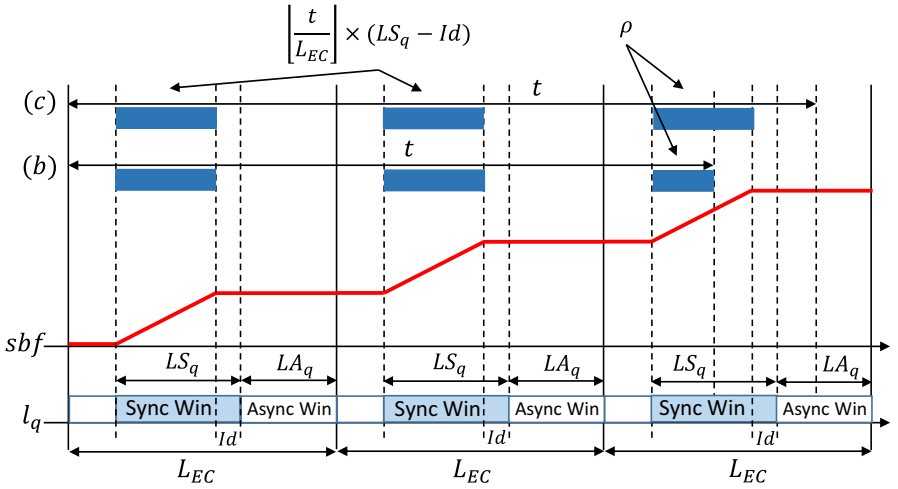


Fig. 14 Supply bound function for the synchronous window

The time interval t in Fig. 14 is calculated using Eq. 26, where ρ is a part of the supply in the last EC and Id_q is the idle time. The calculations for the idle time in the servers will be explained later in this section.

$$sbf_q^{(h)}(t) = \left\lfloor \frac{t}{L_{EC}} \right\rfloor \times (LS_q - Id_q) + \rho \tag{26}$$

We compute the supply in the last EC for the scenarios (b) and (c) as depicted in Fig. 14. If the interval finishes in the middle of the Synchronous Window (i.e., scenario (b)), ρ is computed using Eq. 27. Note that this scenario also covers scenario (a) by using the *max* operation in Eq. 27.

$$\begin{aligned} &\rho = max\left\{t - \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} - (L_{EC} - LS_q - LA_q), 0\right\}, \\ &\text{if } \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} \le t \le \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} + (L_{EC} - LA_q - Id_q) \end{aligned} \tag{27}$$

However, if the time interval finishes after the Synchronous Window but before the end of the EC (i.e., scenario (c)), the calculations for ρ are different as shown in Eq. 28.

$$\begin{aligned} &\rho = max\left\{t - \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} - (L_{EC} - LS_q + Id_q), 0\right\}, \\ &\text{if } t > \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} + (L_{EC} - LA_q - Id_q) \end{aligned} \tag{28}$$

The Asynchronous Window becomes periodically available at known point in time within the EC. Hence, the *sbf* for it is computed in a similar fashion as follows.

$$sbf_q^{(h)}(t) = \left\lfloor \frac{t}{L_{EC}} \right\rfloor \times (LA_q - Id_q) + \rho \tag{29}$$

The resource in the last EC (ρ) is computed for the scenarios using Eqs. 30 and 31.

$$\begin{aligned} &\rho = \max\left\{t - \left\lfloor \frac{t}{L_{EC}} \right\rfloor \times L_{EC} - (L_{EC} - LA_q), 0\right\}, \\ &\text{if } \left\lfloor \frac{t}{L_{EC}} \right\rfloor \times L_{EC} \leq t \leq \left\lfloor \frac{t}{L_{EC}} \right\rfloor \times L_{EC} + (L_{EC} - Id_q) \end{aligned} \tag{30}$$

$$\begin{aligned} &\rho = \max\left\{t - \left\lfloor \frac{t}{L_{EC}} \right\rfloor \times L_{EC} - (L_{EC} - LA_q + Id_q), 0\right\}, \\ &\text{if } t > \left\lfloor \frac{t}{L_{EC}} \right\rfloor \times L_{EC} + (L_{EC} - Id_q) \end{aligned} \tag{31}$$

In order to find the idle time, we derive the maximum message size that crosses the window in link l_q and belongs to $\mathcal{A}^{(h)}$. Equation 32 calculates the idle time, where ω represents the periodic and sporadic activation pattern if the idle time is calculated in the Synchronous and Asynchronous Windows respectively.

$$Id_q = \max_{\substack{\forall m_{z,p} \in \mathcal{A}^{(h)} \wedge\ l_q \in \mathcal{L}_{z,p} \\ \wedge\ AP(m_{z,p}) = \omega}} \{C_{z,p}\} \tag{32}$$

The *rbf* for the servers within the transmission windows is calculated using Eq. 33, where the interference from the higher priority servers is accounted for.

$$rbf_q^{(h)}(t) = \sum_{\forall \mathcal{A}^{(f)} \in \mathcal{S} \wedge\ \Psi_q^{(f)} \leq \Psi_q^{(h)}} \left\lceil \frac{t}{\Pi_q^{(f)}} \right\rceil \Theta_q^{(f)} \tag{33}$$

Finally, the response time of the server is calculated when the *sbf* is equal to or larger than the *rbf*, as shown in Eq. 34.

$$RT_q^{(h)} = min(t > 0) : sbf_q^{(h)} \geq rbf_q^{(h)} \tag{34}$$

The system $\mathcal{S}$ is schedulable if all the servers in the nodes and network links are schedulable. Therefore, the applications should be schedulable in each node and network link. The schedulability of servers can be verified by the existing analysis depending on the scheduler algorithm for the global level scheduler. For instance, if the global level scheduler is the fixed-priority preemptive scheduler according to RM, the response time analysis corresponding to that is used.

6 Reservation design

In this section we address the problem of designing application interfaces, i.e., the end-to-end reservations. The interface design is performed by application developers. Therefore, we consider the problem of designing an end-to-end reservation for one application at a time and, for notational convenience, we drop the application index in this section. The problem of interface design for an application involves designing reservation periods and budgets. The system integrator assigns the priorities of the

reservations such that the reservations are schedulable in the global level. The priority assignment is out of the scope of this paper.

A common practice in designing reservation for a single resource is to compute the bandwidth reservation in which the response times are equal to the corresponding deadlines. In the case of end-to-end resource reservations, however, the problem is more complex. It is not possible to compute each reservation separately because the end-to-end response times depend on all application reservations.

We model the end-to-end reservation design problem as a Constraint Satisfaction Problem (CSP) (Krzysztof 2003). The reservation design using CSP formulation provides separation of concerns in the sense that the problem formulation is independent from the solving technique. To this end, we can use different solving techniques that are provided by the solver and evolved over the time to solve the same problem formulation. This approach is potentially complete and finds the optimum solution provided enough time.

A CSP is a triple of finite set of variables, values and constraints. A CSP solver searches for feasible variable assignments, i.e., a function from the variables to the values such that the constraints are satisfied. The CSP formulation allows considering all reservations simultaneously and designing an end-to-end reservation which is overall efficient. The solution for a CSP is the set of all assignments which satisfy all constraints. A CSP solver performs constraint propagation, branching and search in an intertwined manner. Propagation uses the constraints and removes infeasible values from the value set. For instance, in our case, constraint propagation removes budget values that result in deadline misses. Branching constructs a search tree based on the remaining values in the value set. The search selects a node from the tree that is constructed by branching to be explored. In the following we describe the variables, values and constraints in our CSP formulation.

6.1 Variables

The design problem is to select values for the reservation budgets Θ_j and reservation periods Π_j for all reservations corresponding to an application. Therefore, the variable set of the CSP is as follow:

$$\{\Theta_1, \ldots, \Theta_q, \Pi_1, \ldots, \Pi_q\}.$$

We define the bandwidth of a reservation as follows: $\beta_j = \frac{\Theta_j}{\Pi_j}$. The application footprint γ on the resources is the sum of all reservation bandwidths that belong to the application, i.e. $\gamma = \sum_{j \in R} \beta_j$.

6.2 Values and constraints

We assume that the system designer imposes the minimum reservation periods considering the context switch related overheads and the time resolution of the operating system. Also, we assume that the reservation periods should never exceed the shortest

task/message period within the application. Therefore, we have:

$$\forall j \in R \quad \Pi^{\min} \leq \Pi_j \leq P^{\min}, \tag{35}$$

where $\Pi^{\min}$ represents the minimum allowed period and $P^{\min}$ is the minimum value among all task/message periods within the application. We also assume that the reservation periods are integer multiples of $\Pi^{\min}$.

The reservation budgets cannot exceed the corresponding periods:

$$\forall j \in R \quad \Theta_j \leq \Pi_j. \tag{36}$$

The reservation bandwidth corresponding to the application on the processor resource r_j should be more than the utilization of the tasks assigned to r_j:

$$\beta_j \geq \sum_{\forall i,k} \frac{C_{j,i,k}}{T_{j,i,k}}. \tag{37}$$

We have a similar constraint for the network resource. However, in the case of network resource r_j, the corresponding reservation bandwidth β_j has to be more than or equal to the sum of the bandwidths of all the involved messages:

$$\beta_j \geq \sum_{\forall i,k \mid l_j \in \mathcal{L}_{i,k}} \frac{C_{i,k}}{T_{i,k}}. \tag{38}$$

It is also possible to impose an upper limit for the reservation bandwidth on a particular resource. For instance, when a new application is being developed on top of an existing system, the resources are already allocated to the existing applications. In this case, the new application can only use the remaining slacks on the resources to meet its timing requirements. Therefore, the system integrator may impose maximum allowed bandwidth on the resources for the new application based on the slacks.

The timing requirements of the applications must be respected, i.e., the corresponding deadlines of all transactions have to be respected:

$$\forall \Gamma_i \in \mathcal{A} \quad RT_i \leq D_i \tag{39a}$$

$$Age\_\_Delay_i \leq Age_i \tag{39b}$$

$$Reac\_\_Delay_i \leq Reac_i, \tag{39c}$$

where RT_i, $Age\_\_Delay_i$ and $Reac\_\_Delay_i$ represent the response time, age delay and reaction delay of transaction Γ_i within application $\mathcal{A}$.

6.3 Optimization

The CSP formulation allows to search for the optimal solution with respect to a criterion. In this case the solver uses the branch-and-bound algorithm to prune the search

tree. Basically, instead of evaluating the entire search tree, the solver prunes the nodes that do not improve over the best solution found until the current stage of the search. In this work we consider the following four optimization cases.

6.3.1 Case 1: minimum footprint

Each application occupies a bandwidth (β_j) of the resources in our end-to-end reservation scheme. It is desirable to reserve minimum bandwidths for the applications to allow integration of more applications on the same underlying resources. Therefore, the optimization objective is:

$$\begin{aligned} &\text{Minimize: } \gamma = \sum_{j \in R} \beta_j, \\ &\text{Subject to: (35) (36) (37) (38) (39a) (39b) (39c).} \end{aligned}$$

In this case, we assume that the system integrator has not imposed any upper bound on the allowed reservation bandwidths while the application under analysis has deadlines for response times, age delays and reaction delays. Then, the objective is to minimize the application footprint.

6.3.2 Case 2: best performance

In this case the system integrator has imposed an upper bound on the application footprint γ and the application designer is interested in finding a design in which the application performance is maximized, i.e., the response times, age delays or reaction delays are minimized. For instance, if the response times are of interest, the optimization objective is as follows:

$$\begin{aligned} &\text{Minimize: } \sum_{\forall \Gamma_i \in \mathcal{A}} RT_i, \\ &\text{Subject to: (35) (36) (37) (38) (39b) (39c),} \\ &\qquad \gamma \leq \text{maximum allowed bandwidth.} \end{aligned}$$

When multiple objectives are of interest for optimizations, then we combine the criteria in one objective function and optimize based on a new criterion which is a weighted sum of all the criteria.

6.3.3 Case 3: minimum overhead

Another possible target is to minimize the overhead imposed by the reservations management mechanism in processor resources, namely their periodic activation and associated context switching. In such cases we can optimize for the following objective:

$$\text{Maximize: } \sum_{\forall j \in R_{cpu}} \Pi_j,$$
$$\text{Subject to: (35) (36) (37) (38) (39a) (39b) (39c),}$$

where, R_{cpu} is the set of all processor resources of the application.

6.3.4 Case 4: combined footprint and performance

It is also interesting to optimize with respect to multiple objectives. In this case, we use a linear combination of different criteria as the optimization objective. For instance, the application footprint and performance are two contradicting objectives since larger footprints usually result in designs with better performance than small footprint. Therefore, the goal is to combine these two objectives and search for designs which have smaller footprints as well as good performance.

$$\text{Minimize: } w_\gamma \gamma + \sum_{\forall \Gamma_i \in \mathcal{A}} w_i RT_i,$$
$$\text{Subject to: (35) (36) (37) (38) (39a) (39b) (39c),}$$

where w_γ and w_i are the weights associated to the footprint and response time objectives respectively. These weights are used to adjust the trade off among different objectives.

6.4 Design tool

We have developed an open-source C++ exploration tool which implements the above CSP formulation.[4] The set of tasks and messages as well as the set of transactions are provided as inputs to the tool in XML format. The users can select the optimization criterion by specifying it in a setting file. We have used the Gecode toolkit (Schulte et al. 2015) for modeling and solving the CSP. The analysis presented in Sect. 5 should be encoded in our constraint model, however, this analysis is very complex and cannot be expressed using the standard constraints provided by Gecode. To this end, we have implemented a new constraint propagator which implements the schedulability analysis.

6.4.1 Discussions

Throughout our experiments we have observed that the optimal designs are usually found quickly for the first optimization case, i.e., when optimizing for minimum footprint. Optimizing for performance or overhead, however, usually takes longer time. This is because a larger search space is often needed to be explored in these cases, therefore, the solver needs longer time for searching the entire space. In the cases

[4] The tool is available at: https://github.com/nimazad/e2e-res.

where acquiring optimal designs require long time, we can provide a timeout to the solver and stop the search at an intermediate point. The result of the search may not be the global optimum, though, as the search is terminated before the exploration of the entire space. However, the result is still a feasible solution.

Recall that the CSP solver uses a branching strategy to construct the search tree. Gecode allows enforcing specific branching strategies, i.e., one can specify which variables should be branched first and which values from the domain of the variables should be selected for the assignments. For instance, we can configure the solver to branch based on the periods first and on the budgets second. Furthermore, we can enforce that, for instance, when branching on the periods the solver should first try assigning the minimum values of the domains. The branching strategies used in our tool depends on the selected optimization objective. In other words, we always start the search from promising parts of the search space. To this end, even in the cases where the search does not complete before the timeout, there is a high probability that the obtained design is optimal or near optimal. However, the solver needs longer time for verification.

7 Case study and evaluation

In order to show the applicability of the proposed model, resource reservation design method and end-to-end timing analysis we present a case study from the vehicular systems domain. We consider an Autonomous Steering Control (ASC) system, that provides electronic steer control to a vehicle using mechanical and electronic components. The ASC system uses an Ethernet network for communication as the backbone network in the vehicle. The architecture of the ASC system is depicted in Fig. 15. The ASC system is distributed over six Electronic Control Units (ECUs) and two HaRTES switches. There are four Wheel Control (WC) ECUs; one Steer Control (SC) ECU; and one Collision Avoidance Control (CAC) ECU. Moreover, there is a camera (CAM) that is mounted in front of the vehicle to gather the video frames for collision avoidance purpose.

The ASC system is divided into two applications: (i) the Collision Avoidance (CA) application, and (ii) a conventional steering control, known as Steer-By-Wire (SBW) application. Each application together with its transactions is described in the next subsection. It should be noted that the parameters for both applications are inspired by an industrial use case from our industrial partners. All experiments related to the reservation designs are performed on an Intel Core i7-5500U CPU clocked at 2.4 GHz with two cores and 8 GB memory.

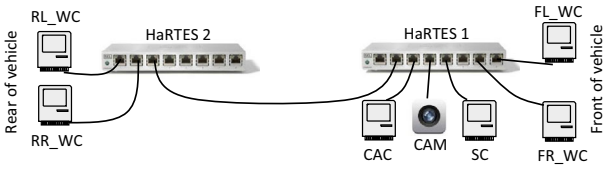


Fig. 15 The network architecture for the ASC system

7.1 Collision avoidance application

The main function of this application is to detect obstacles by means of information from a radar and video frames from a camera to calculate a new steering angle accordingly. This information is collected in the CAC ECU. The CAC ECU sends this information in an Ethernet message to the SC ECU which is responsible for computing the steer actuator signals accordingly. The transactions belonging to this application are illustrated in Fig. 16. In the above transactions, the WCETs of the tasks are selected among 100, 200 and 400 μs depending on the task function. The notation below each task and message shows its activation pattern, which is independent (denoted by I) and periodic (denoted by P) with period of 40 ms in all tasks and messages. The execution time of the tasks and the size of messages are also denoted below each task. The message payload size for the control and radar signals is 64 bytes, whereas the message payload size for the video frame is 1500 bytes. For these transactions, the end-to-end deadline is 100 ms, while the age and reaction constraints are 110 and 150 ms, respectively.

7.1.1 Designing reservations

In the following we design different end-to-end reservations for the collision avoidance application corresponding to different optimization cases presented in Sect. 6.

Minimum footprint The search for the minimum footprint design for the collision avoidance application completes in 1.5 s. This design is presented in Table 1. This design uses 7% of the system resources since each resource uses 1%. The tool sets the minimum allowed periods (1 ms) to all reservation periods. The performance of this design is presented in Table 2 which shows that all the timing constraints, specified on the transactions, are satisfied.

Best performance We limit the overall footprint of the application to different values from 10 to 60% and run the tool six times to obtain the designs with the best response times. Note that the optimization criterion in this case is to minimize the sum

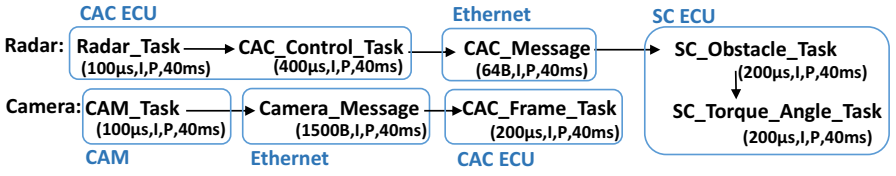


Fig. 16 Transactions in the CA application

Table 1 Minimum footprint design for the collision avoidance application

	CAC ECU	SC ECU	CAM	CAC ECU	l_j
α_j (%)	1	1	1	1	1
Θ_j (μs)	10	10	10	10	10
Π_j (ms)	1	1	1	1	1

Table 2 Performance in minimum footprint design for the collision avoidance application

	RT	Age delay	Reaction delay
Radar (μs)	53,940	90,990	130,990
Camera (μs)	47,960	50,990	90,990

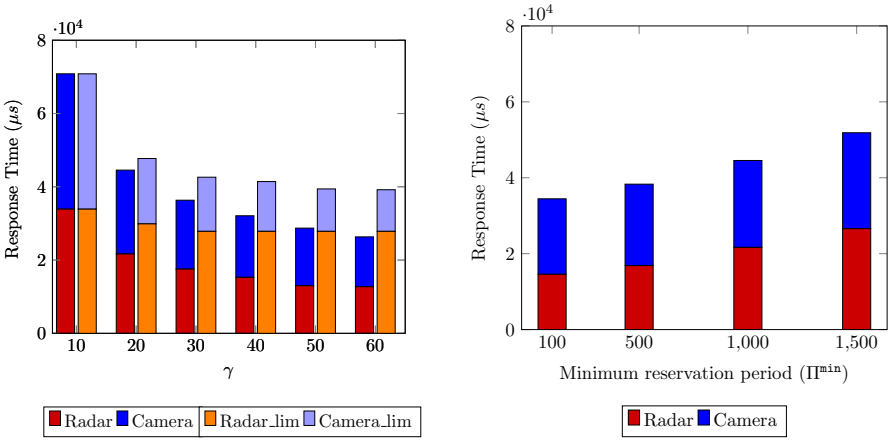


Fig. 17 Sum of the transaction response times against (i) the overall footprint where $\Pi^{\min} = 1$ ms (*left figure*) and (ii) the minimum reservation periods where $\gamma = 20\%$ (*right figure*)

of the response times of the two transactions within the application. Figure 17 shows the sum of the response times against the maximum allowed footprint. The figure shows that the reduction in response times is more significant when increasing γ from 10 to 20% than increasing it to the values that are above 20%. Also, we have observed that by providing 8.5 times more resources (from 7 to 60%) we can achieve around 74% better performance in terms of response times. The age delays and reaction delays also follow a similar trend. In another experiment, we have modified the minimum allowed period for the reservations ($\Pi^{\min}$), which sets the time resolution of the reservation periods, and searches for the best performance design. The maximum allowed footprint is set to 20% in these cases. The results are depicted in Fig. 17. The figure shows that larger $\Pi^{\min}$ results in worse performance. In all of the above experiments we have limited the search time to 30 min. Note that in all of the cases, the tool finds the first design in less than 2 s. A better design is found a few seconds after the first design in some cases, while the attempt to improve the first two designs in the rest of the exploration time fails in most cases.

So far we have only imposed a limit on the total application footprint. In the case where an application is being integrated to a system which already hosts other applications, we may have limited availability of particular resources. We perform a new experiment to evaluate the reservation design tool. In this experiment we assume that there are only 2% of the resources that are available to CAC ECU and SC ECU. We repeat the search for best response times assuming different total footprints and considering the above constraints on CAC ECU and SC ECU. The results are depicted in

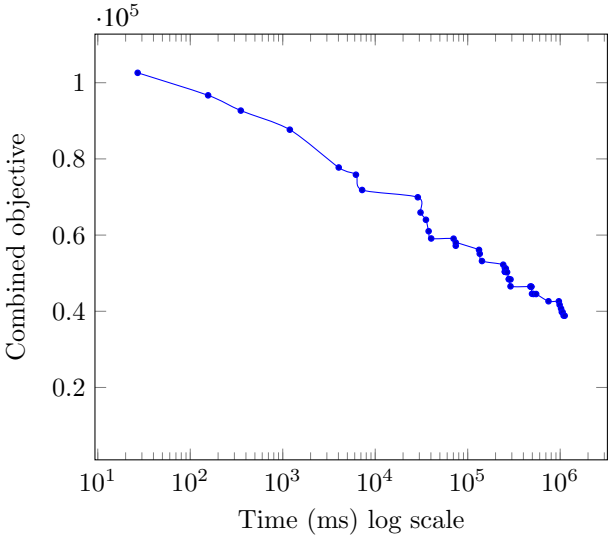


Fig. 18 Evolution of the design's objective value against search time

Fig. 17 using the captions Radar_lim and Camera_lim. Increasing the total allowed footprint from 30 to 60% does not have significant effect on the response times as availability of the two resources is limited.

Minimum overhead The search for the minimum overhead design with $\gamma = 7$ and 30 min timeout results in the following reservation periods: $\Pi_{\text{CAC ECU}} = 10$ ms, $\Pi_{\text{SC ECU}} = 15$ ms and $\Pi_{\text{CAM}} = 20$ ms. This set of end-to-end reservations imposes 15 times less overhead on the processor resources than the above designs. On the other hand, the performance of this design is around 44% worse than the performance of the minimum footprint design that is reported in Table 1.

Combined footprint and response times In another experiment, we search for the optimal design with respect to the combined footprint and response times. In this experiment, we assume $w_\gamma = 1000$ and $w_i = 1$. This weight assignment indicates that we consider 1% footprint improvement to be equivalent to 1 s improvement in response time. Similar to the previous cases we use 30 min timeout. Figure 18 shows the quality of obtained design against the amount of time spent on searching. The figure shows that the improvements in the first few seconds are more significant than the rest of the search duration. Note that the x-axis is in the logarithmic scale. The final design is found after around 9 min and it uses 20% of the system resources, i.e. $\gamma = 20\%$.

7.2 Steer-by-wire application

Each WC ECU gathers the wheel information including the angle and torque of the wheel. This information is sent to the SC ECU via an Ethernet message. Apart from this message, the SC ECU receives several sensor values including steering angle,

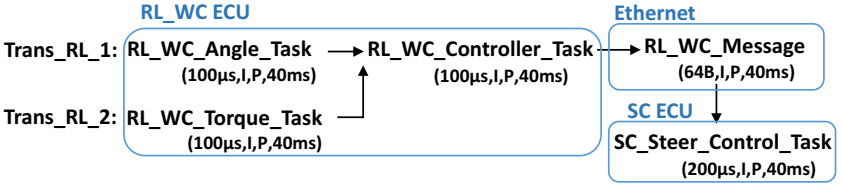


Fig. 19 Transactions from the rear-left WC ECU to the SC ECU

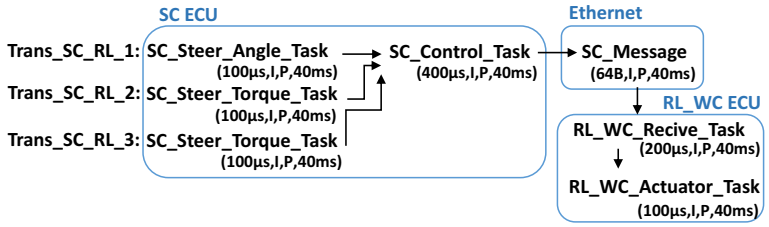


Fig. 20 Transactions from the SC ECU to the rear-left WC ECU

steering torque and vehicle speed. Based on this information, the SC ECU provides feedback on steering torque to the feedback torque actuator. This actuator provides the feeling of turning effect for the driver. Moreover, the SC ECU sends an Ethernet message to each WC ECU. This message includes the steer angle and torque signals which are used by the WC ECUs to control the wheels actuators. For the sake of clarity and better readability, we only show the transactions for one wheel, i.e., rear-left (RL) wheel. Two transactions from the RL_WC ECU to the SC ECU are depicted in Fig. 19. Note that the transactions share the RL_WC_Controller_Task (in the RL_WC ECU), RL_WC_Message (in the network) and SC_Steer_Control_Task (in the SC_ECU). Similarly to the previous application, the notation below each task and message shows its activation pattern, execution time of tasks and size of messages, e.g., (100 μs, I, P, 40 ms) shows that the task is periodically activated by an independent source with a period of 40 ms and execution time of 100 μs.

In addition, there are three transactions to send information from the SC ECU to WC ECU. These transactions also share tasks and messages which are illustrated in Fig. 20. Since there are five transactions for each wheel, the total number of transactions in the SBW application is equal to 20. The constraints specified on the transactions include the deadline (100 ms), age constraint (100 ms) and reaction constraint (140 ms).

7.2.1 Designing reservations

The search for minimum footprint reservations for the steer-by-wire application ends after 126 s. The search time for the steer-by-wire application is significantly larger as compared to the collision avoidance application because of large size of the former application (20 transactions). The result of the minimum footprint design is presented in Table 3, while Table 4 presents the performance figures for this design. We also run the tool for best performance design with different imposed maximum footprints. We run the search for 30 min for each given γ. The result is presented in Fig. 21.

Table 3 Minimum footprint design for the collision avoidance application

	WC ECUs	SC ECU	l_4	Other links
α_i (%)	4	20	2	1
Θ_i (μs)	40	200	20	10
Π_i (ms)	1	1	1	1

Table 4 Performance in minimum footprint design for the collision avoidance application

	RT	Age delay	Reaction delay
Trans_RL_1 (μs)	20,570	81,800	121,800
Trans_RL_2 (μs)	20,570	81,800	121,800
Trans_SC_RL_1 (μs)	23,250	83,940	121,800
Trans_SC_RL_2 (μs)	23,250	83,940	121,800
Trans_SC_RL_3 (μs)	23,250	83,940	121,800

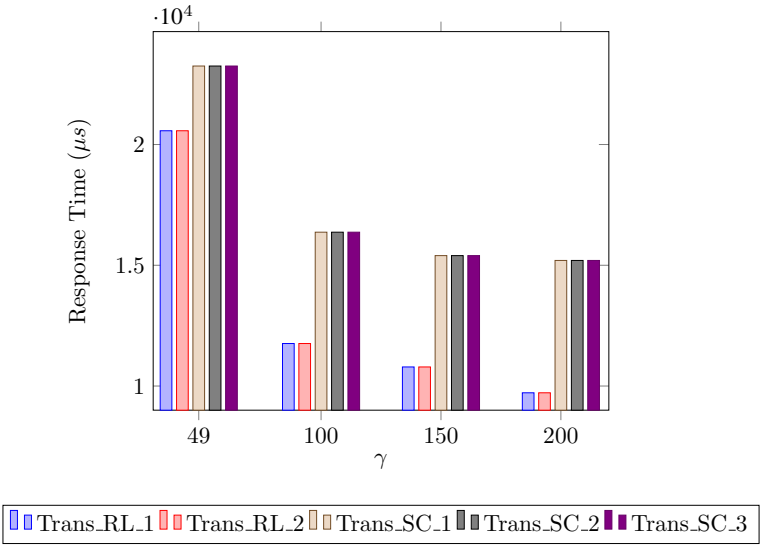


Fig. 21 Response times against footprint γ for the steer-by-wire application

Note that the y-axis, i.e. response time, starts from 9 ms in this figure. In these cases, the first design is usually found under 300 s. The difference between the performance of design with $\gamma = 100\%$ and design with $\gamma = 49\%$ is larger than the other cases. Finally, we run the tool for combined footprint and response times objective with the same weight values as for the collision avoidance application and 30 min timeout. The results are shown in Fig. 22. In comparison to the collision avoidance application, the quality of the designs evolve slower in this case as the application consists of more transactions.

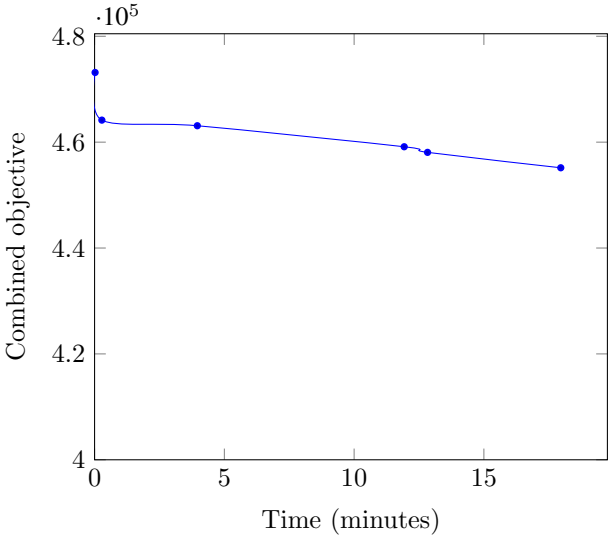


Fig. 22 Evolution of the design's objective value against search time

7.3 Discussion

In this case study we have demonstrated the usability of the proposed end-to-end resource reservation model. The case study, inspired from a real industrial application, is of reasonable size as it consists of two applications each with several transactions. The resource reservation parameters computed by the proposed reservation design method render the system schedulable. We have also demonstrated an important feature that is offered by the proposed model, method and analysis which allows the applications to be designed and analyzed in isolation without being affected by each other. This feature can assist the developers and integrators at the design phase by supporting the design and implementation of each application by a separate team. Moreover, to improve the model and to guide the designers for assigning the best reservation parameters, a reservation design method is presented. We have also shown that this method can find the reservation design for heavier applications, consisting of more than 20 transactions, in reasonable amount of time. We showed that our reservation design method always finds a feasible solution after a few seconds. Finding the optimal solution, however, depends on the selected optimization criterion. Minimum footprint designs are also found quickly, while the best performance designs usually require search over large portions of the search space, hence take longer time. However, since we guide the solver in such a way that it searches promising parts of the search space first, we quickly achieve good designs.

8 Conclusion and future work

In this paper we have proposed a new model for end-to-end resource reservations in distributed embedded systems. The model supports reservation on both computation and communication resources. The model allows reservation on any network protocol that supports a reservation mechanism including the AVB and HaRTES Ethernet networks. We have also considered a general transactional model such that the transactions can have several tasks in each node and several messages in the network. In addition, the tasks and messages in the transaction can have various activation patterns including trigger, data and mixed chains. Such activations patterns are common in several industrial domains. Therefore, the presented model responds to a vast number of industrial domains with different requirements with respect to design choices. We have presented end-to-end timing analysis that supports end-to-end resource reservations. The analysis computes the end-to-end response times as well as end-to-end delays, including the data age and reaction delays, for each transaction. We have also identified that the existing end-to-end delay analysis does not support those real-time networks that can autonomously initiate transmissions. We have provided a solution that allows the existing analysis (without resource reservations) as well as the analysis presented in this paper (with resource reservations) to be applicable to such networks. We have also presented a method to design end-to-end resource reservation parameters. The design method exploits several optimization criteria in computing the reservation parameters. The design method can play a key role in helping the system integrator making efficient design decisions and improving the schedulability of the system. Finally, we have shown the usability of the proposed model, reservation design method and end-to-end timing analysis with the help of an automotive case study. In the future, we plan to transfer the proposed model, method and analysis to the industry by implementing them in an existing industrial component model and a tool suite, which is called Rubus-ICE. The tool supports model- and component-based development of vehicle software. Then, we are planning to perform several tests on a hardware platform.

Acknowledgements The research leading to this paper has been supported by the Swedish Foundation for Strategic Research (SSF) within the projects PRESS and FIC, the Swedish Knowledge Foundation (KKS) within the project PreView, and the EU integrated project CONTREX (FP7-611146). We would like to thank all our industrial partners, specially Arcticus Systems, Volvo Construction Equipment and BAE Systems Hägglunds, Sweden.

References

Almeida L, Pedreiras P (2004) Scheduling within temporal partitions: response-time analysis and server design. In: Proceedings of the 4th ACM international conference on embedded software

Aminifar A, Bini E, Eles P, Peng Z (2016) Analysis and design of real-time servers for control applications. IEEE Trans. Comput. 65(3):834–846

Ashjaei M, Behnam M, Pedreiras P, Bril RJ, Almeida L, Nolte T (2014) Reduced buffering solution for multi-hop HaRTES switched Ethernet networks. In: The 20th IEEE international conference on embedded and real-time computing systems and applications

Ashjaei M, Mubeen S, Behnam M, Almeida L, Nolte T (2016) End-to-end resource reservations in distributed embedded systems. In: IEEE 22nd international conference on embedded and real-time computing systems and applications (RTCSA), pp 1–11

Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A (2003) Xen and the art of virtualization. In: ACM symposium on operating systems principles

Chatterjee S, Strosnider J (1995) Distributed pipeline scheduling: end-to-end analysis of heterogeneous, multi-resource real-time systems. In: The 15th international conferences on distributed computing systems

Cucinotta T, Palopoli L (2010) QoS control for pipelines of tasks using multiple resources. IEEE Trans Comput 59:416–430

Davis R, Burns A (2008) An investigation into server parameter selection for hierarchical fixed priority pre-emptive systems. In: 16th international conference on real-time and network systems

Deng Z, Liu JWS (1997) Scheduling real-time applications in an open environment. In: The 18th IEEE real-time systems symposium

Dermler G, Fiederer W, Barth I, Rothermel K (1996) A negotiation and resource reservation protocol (NRP) for configurable multimedia applications. In: The third IEEE international conference on multimedia computing and systems

Easwaran A, Anand M, Lee I (2007) Compositional analysis framework using EDP resource models. In: 28th IEEE international real-time systems symposium

Feiertag N, Richter K, Nordlander J, Jonsson J (2008) A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics. In: Workshop on compositional theory and technology for real-time embedded systems

Feng XA (2005) Towards real-time enabled microsoft windows. In: The 5th ACM international conference on embedded software

Fisher N, Dewan F (2009) Approximate bandwidth allocation for compositional real-time systems. In: 21st Euromicro conference on real-time systems

FlexRay Consortium (nd) FlexRay communications system protocol specification, Ver. 3.0.1, Oct, 2010

Ghosh S, Hansen J, Rajkumar RR, Lehoczky J (2004) Integrated resource management and scheduling with multi-resource constraints. In: The 25th IEEE international real-time systems symposium

IEEE (2014) IEEE Std. 802.1Q, IEEE standard for local and metropolitan area networks, bridges and bridged networks

ISO 11898-1 (nd) Road vehicles interchange of digital information controller area network (CAN) for high-speed communication, ISO Standard-11898, Nov 1993

Krzysztof RA (2003) Principles of constraint programming. Cambridge University Press, Cambridge

Lakshmanan K, Rajkumar R (2008) Distributed resource kernels: OS support for end-to-end resource isolation. In: IEEE real-time and embedded technology and applications symposium

Lipari G, Bini E (2003) Resource partitioning among real-time applications. In: Proceedings of 15th Euromicro conference on real-time systems

Lipari G, Bini E (2005) A methodology for designing hierarchical scheduling systems. J Embed Comput 1:257–269

Lorente JL, Lipari G, Bini E (2006) A hierarchical scheduling model for component-based real-time systems. In: Proceedings 20th IEEE international parallel distributed processing symposium

Mubeen S, Mäki-Turja J, Sjödin M (2013) Support for end-to-end response-time and delay analysis in the industrial tool suite: issues, experiences and a case study. Comput Sci Inform Syst 10(1):453–482

Nolte T, Nolin M, Hansson HA (2005) Real-time server-based communication with CAN. IEEE Trans Ind Inform 1(3):192–201

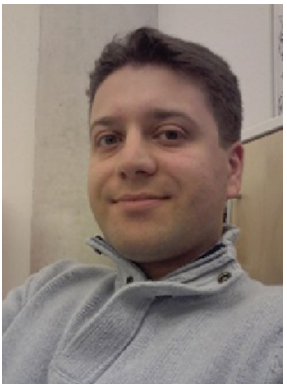
Oliveira AB, Azim A, Fischmeister S, Marau R, Almeida L (2014) D-RES: correct transitive distributed service sharing. In: IEEE emerging technology and factory automation

Rajkumar R, Lee C, Lehoczky J, Siewiorek D (1997) A resource allocation model for QoS management. In: The 18th IEEE real-time systems symposium

Saewong S, Rajkumar R, Lehoczky JP, Klein MH (2002) Analysis of hierarchical fixed-priority scheduling. In: Proceedings of the 14th Euromicro conference on real-time systems

Santos R, Behnam M, Nolte T, Pedreiras P, Almeida L (2011) Multi-level hierarchical scheduling in Ethernet switches. In: Proceedings of the international conference on embedded software

Schulte C, Tack G, Lagerkvist MZ (2015) Modeling and programming with Gecode. Technical report
Shin I, Lee I (2003) Periodic resource model for compositional real-time guarantees. In: 24th IEEE real-time systems symp
Sojka M, Píša P, Faggioli D, Cucinotta T, Checconi F, Hanzálek Z, Lipari G (2011) Modular software architecture for flexible reservation mechanisms on heterogeneous resources. J Syst Archit 57:366–382
Sprunt B, Sha L, Lehoczky J (1989) Aperiodic task scheduling for hard-real-time systems. Real-Time Syst J 1(1):27–60
Strosnider JK, Lehoczky JP, Sha L (1995) The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. IEEE Trans Comput 44(1):73–91
Wandeler E, Thiele L (2005) Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling. In: The 5th ACM international conference on embedded software, EMSOFT '05



Mohammad Ashjaei has received his Ph.D. degree in Computer Science and Engineering from the same university in November 2016. He was a visiting researcher at University of Aveiro, Portugal, for one month in 2013. His main research interests are real-time distributed systems, scheduling algorithms, resource management and reservation. Moreover, he is interested in cloud computing, reservation mechanisms in cloud computing and admission controls.

Nima Khalilzad received his B.Sc. in Software Engineering from Arak University, Iran, in 2009. He obtained his M.Sc. and Ph.D. in Computer Science and Engineering from Mälardalen University, Sweden in 2011 and 2015 respectively. He has been a visiting researcher at McGill University, Montreal, Canada for five months in 2013 and 2014. He has been a postdoc researcher at KTH Royal Institute of Technology, Sweden from 2015 to 2017. His research interests are reservation-based real-time scheduling, feedback scheduling and design space exploration using constraint programming and evolutionary metaheuristics.

Saad Mubeen is a Senior Member of IEEE. He is a Senior Lecturer (Assistant Professor) at the School of Innovation, Design and Engineering at Mälardalen University, Sweden. He is also working in the automotive industry, where he is employed by Arcticus Systems AB, Sweden. He has formerly worked as consultant for Volvo Construction Equipment, Sweden. He received his Ph.D. in Computer Science and Engineering from Mälardalen University in 2014. His research interests include model-based development of vehicular embedded systems with a focus on timing models, end-to-end timing analysis and multi-core platforms. He has coauthored over 90 research publications in international peer-reviewed journals, conferences, workshops and book chapters.

Moris Behnam has awarded a B.Eng., and M.Sc. in Computer and Control Engineering at the University of Technology, Iraq, and also M.Sc., Licentiate, and Ph.D. in Computer Science and Engineering at MDH, Sweden, in 1995, 1998, 2005, 2008 and 2010 respectively. He has been a visiting researcher at Wayne State University, USA in 2009 and he has been a Postdoctoral Researcher at University of Porto in 2011. His research interests include real-time scheduling, synchronization protocols, multicore/multiprocessor systems, distributed embedded real-time systems and using control theories in real-time scheduling.

Ingo Sander received the M.Sc. degree in Electrical Engineering from the Technical University of Braunschweig, Germany, in 1990 and the Ph.D. degree and docent degree from KTH Royal Institute of Technology, Sweden, in 2003 and 2009, respectively. Between 1991 and 1993 he worked as system design engineer at Ericsson, Sweden. In 1993 he joined KTH, where he since 2005 holds a position as associate professor in Electronic System Design. His main research interests are located in the area of design methodologies for embedded systems including system modelling, design space exploration and system synthesis.

Luis Almeida graduated in Electronics and Telecommunications Eng. in 1988 and received a Ph.D. in Electrical Eng. in 1999, both from the University of Aveiro in Portugal. He is currently an associate professor in the Electrical and Computer Engineering Department of the University of Porto (UP), Portugal, and a senior researcher in the Telecommunications Institute at UP where he coordinates the Distributed and Real-Time Embedded Systems (DaRTES) lab. Among several appointments, he was a member of the IEEE Technical Committee on Real-Time Systems (2008–2013), Program and General Chair of the IEEE Real-Time Systems Symposium (2011–2012 respectively) and Vice-President of the RoboCup Federation (2011–2013) being Trustee of this organization since 2008. His research interests include those related to real-time networks for distributed industrial/embedded systems including for teams of mobile robots.

Thomas Nolte was awarded a B.Eng., M.Sc., Licentiate, and Ph.D. degree in Computer Engineering from Mälardalen University (MDH), Västerås, Sweden, in 2001, 2002, 2003, and 2006, respectively. He has been a Visiting Researcher at University of California, Irvine (UCI), Los Angeles, USA, in 2002, and a Visiting Researcher at University of Catania, Italy, in 2005. He has been a Postdoctoral Researcher at University of Catania in 2006, and at MDH in 2006–2007. He became an Assistant Professor at MDH in 2008, and Associate Professor at MDH in 2009. In 2012 he became Full Professor of Computer Science.