Adaptive Multi-Resource End-to-End Reservations for Component-Based Distributed Real-Time Systems

Nima Khalilzad[1], Mohammad Ashjaei[1], Luis Almeida[1,2], Moris Behnam[1], Thomas Nolte[1]
[1] MRTC/Mälardalen University, Sweden
[2] IT/DEEC/University of Porto, Portugal
nima.m.khalilzad@mdh.se

***Abstract*—Complexity in the real-time embedded software domain has been growing rapidly. The component-based software development approach facilitates the development process of such software systems by dividing a complex system into a number of simpler components. Resource reservation techniques have been widely used for providing resources to real-time software components. In this paper we target real-time components operating on a distributed resource infrastructure. Furthermore, we target a class of software components which demonstrate dynamic resource consumption behavior. A prime example of such components is a multimedia software component. In the paper, we present a framework supporting multi-resource end-to-end resource reservations. We reserve resource bandwidths on both processor resources as well as on the network resources. The proposed framework utilizes a Multiple Input Multiple Output (MIMO) controller which adjusts the sizes of reservations tracking the dynamic resource demands of the software components. Finally, we present a case study using a multimedia component to demonstrate the performance and efficiency of our framework.**

I. Introduction

Complex distributed systems are currently disseminated over a large range of application domains, particularly inherent in cyber-physical/embedded systems. These systems are typically subject to several non-functional constraints, stretching from resource limitations to timeliness, including safety and other constraints. Taming complexity in their design is particularly important to ensure a swift development and a correct software product.

To this end, component-based software development approaches are particularly well suited. In such approaches a complex software system is divided into a number of simpler software components. Each component is developed independently, potentially by different teams. The components are integrated at the final stage of the development. On the other hand, for predictable resource provisioning, we can reserve a fraction of the resource time for each component. This technique is known as the resource reservation technique [1], [2], [3]. Moreover, reservations enforce mutual isolation, particularly temporal isolation. Thus, components that run inside adequate reservations can be proved correct independently of other components possibly running in the system.

However, beyond correctness, current design trends aim at resource efficiency, reducing the component footprint over the set of needed resources, particularly computing and communication resources, and changing the resource requirements at run time according to instantaneous needs. In this scope, dynamic reservations can provide a suitable solution to guarantee a continued adequate match between the varying resource requirements and the provided reservations. Dynamic reservation schemes are of particular interest in the multimedia applications in which the instantaneous resource requirements are highly dynamic. Dynamic reservations have been mostly studied for single resource systems (e.g. [4], [5]). Distributed dynamic reservations taken in a holistic way, particularly considering processor and network resources in an integrated fashion, have not received much attention. These, however, are necessary for common multimedia systems ranging from area surveillance to process monitoring and even safety driver assistance.

This paper aims at contributing a solution to such a problem, making use of dynamic resource reservations on processor and network resources, coupled by dynamic component requirements. We provide a solution which allows for matching of dynamic requirements with the resource reservations, reducing typical overprovisioning of static designs (i.e., designed with fixed reservations) and thus resource usage. In turn, since more resource capacity will be potentially available, the system service will also be improved, e.g. allowing for serving more components and/or with more quality.

In this paper, we consider a component model in which each component contains multiple tasks spread over a distributed system that communicate through the network using messages. We consider an end-to-end soft real-time model. We reserve a fraction of the processor as well as network resources needed by the component to satisfy its timing requirements. We refer to this reservation scheme as *multi-resource end-to-end reservations*. Furthermore, we continually monitor the actual resource usages of the components, and we adjust the reservation sizes to match their instantaneous resource requirements. In particular, we present the following contributions in this paper:

- We present a new framework featuring multi-resource end-to-end reservations in which the reservation sizes are adaptive;
- We design a Multiple Input Multiple Output (MIMO) Linear Quadratic Regulator (LQR), which adjusts the reservation sizes during run-time;
- We present an on-line system identification method

The research leading to this paper has received funding from the Swedish Research Council (Vetenskapsrådet) under the project ARROWS, the Swedish Foundation for Strategic Research (SSF) via the research project PRESS and the Swedish Knowledge Foundation through the project RV-RED.

based on Recursive Least Squares (RLS), which identifies the dynamics of the resource requirements;

- We present a surveillance case study in which three processor nodes are used, connected through an Ethernet switch.

The rest of the paper is organized as follows. In the next section we review the related work regarding reservation techniques on processors, networks and distributed systems. Section III presents our modeling approach with respect to the resources and components. The architecture of our framework is presented in this Section IV. We present the control design as well as the system identification method in Section V. We present a surveillance component case study in Section VI. Finally, we conclude the paper in Section VII where we also describe the future directions.

II. Related work

In the following, we review the reservation techniques inherent in three different areas, processor resources, network resources, and end-to-end resources in distributed systems. We also review two groups of works: (i) static reservations, and (ii) dynamic reservations. From a modeling perspective, several resource models have been proposed for modeling resource reservation techniques. For instance, the Periodic Resource (PR) model [1] uses a period and a budget for characterizing a reservation. A reservation is guaranteed to receive a specific budget during each time interval equal to the period. The budget is reduced while the resource is consumed by a particular component, and it is replenished at the start of the period.

Resource reservations on processors. A number of resource reservation models are realized on the processor resources. For instance, the Constant Bandwidth Servers (CBS) [6] are implemented in the Linux kernel [7], or the PR model is implemented in VxWorks [8]. When the tasks have dynamic resource demands, it is desirable to adapt the reservation parameters to deal with the resource demand changes. Adaptive CBS is promoted in the AQuoSA project [4] for dynamic tasks such as video decoders. The ACTORS projects [9] uses adaptive CBS on multiprocessor platforms. In [5], the budget of periodic servers are adapted tracking the processor demand of the components. In this work the model is hierarchical, i.e., the periodic servers may contain multiple tasks as well as multiple child periodic servers.

Resource reservations on network. The same modeling concepts as in processors have been applied on the network resources. A general category of the resource management in network is traffic shapers [10]. The purpose of these shapers is to limit the amount of traffic that a node submits to the network in a given time interval. Similar to the techniques used by processor servers, the traffic shapers use methods based on capacity which is replenished with different policies, e.g. credit-based shaping in Ethernet AVB [11]. Moreover, some real-time Ethernet protocols enforce a cyclic-based transmission and reserve windows for different classes of traffic (e.g., Ethernet POWERLINK [12], FTT-SE [13] and HaRTES [14]). Also, a hierarchical server model [15] is proposed for the Ethernet switches in the context of the FTT-SE protocol to reserve a portion of bandwidth for different traffic types, hence providing temporal isolation among them. An online QoS management [16] is proposed in the context of a multimedia real-time application, which adapts the video compression parameters and the network bandwidth reservations to provide the best possible QoS to the streams. Our end-to-end reservation framework can use the above network technologies for reserving the network resources.

Registering resource reservations on network. In order to reserve resources for streams in the network several protocols have been proposed, where they use similar concepts. For instance, Stream Reservation Protocol (SRP) [17] defines a set of procedures to reserve network resources for the specific traffic streams, which are crossing through an Ethernet Audio Video Bridging (AVB) network. The SRP protocol forces the traffic to be registered on the AVB switches through its path, before being transmitted. Furthermore, a Resource ReSerVation Protocol (RSVP) [18] was proposed to reserve resources for a stream with a specific Quality of Service (QoS) requirement. This protocol operates using an admission control, which checks whether there are enough resources to supply the requested QoS requirement. In both protocols, the mechanism performs by sending a request through the network and checking in each node the availability of resources. These protocols provide a support for communicating new reservations in adaptive reservation schemes such as ours.

Resource reservations in distributed systems. Few authors have addressed the end-to-end reservation of resources for distributed systems, including processor and network resources. A distributed kernel framework with a resource manager in each node has been designed and implemented to provide an end-to-end timeliness guarantee [19]. Also, a resource management system, called D-RES [20], has been developed to handle shared resources among multiple applications in distributed systems. A very close work related to ours is the one presented in [21] in which a pipeline task is considered. Tasks may use one of the resources available in the system to carry on their computations. Adaptive CBS is used to track the resource demand of the tasks. In addition, a general model, called Q-RAM [22], has been developed to manage the resources shared among multiple applications. The applications in this framework have different operation levels with different qualities depending on the available resources. However, they have to satisfy their needs such as timeliness, reliability and data quality. The model allocates the resources to the applications considering that the overall system utility becomes maximum while the applications meet their minimum needs. This model has been extended in [23] for the systems with rapidly changing resource usage.

The main difference of our work with [21] and [22] is that we consider adaptation for components which may in turn be composed of multiple tasks. The existence of multiple tasks inside one component makes the system dynamics model in those works inapplicable to our setting. Therefore, we use an on-line model identification method for estimating the parameters of the model. Besides, we perform the adaptations in an integrated fashion for all resources of the component using MIMO controllers. This is because the MIMO control approach allows us to simultaneously adapt the bandwidths of all reservations considering the possible coupling among them. Finally, in our framework we explicitly consider network resources, and we present the result of our case study in which we used a common network technology.

Adaptive distributed systems. Feedback scheduling techniques have been used in the context of distributed systems. In particular a line of work in this context focuses on keeping the utilization of the processors below their schedulability threshold. For instance Stankovic *et al.* have studied this problem for independent tasks [24]. On the other hand, the following two frameworks are proposed for end-to-end task models: EUCON [25] and DEUCON [26]. While EUCON uses a centralized controller, DEUCON employs a decentralized approach in which task rates (periods) are adapted using model predictive controllers. The main difference of our paper with the aforementioned works is the following. Since we consider component-based systems in which a component is comprised of a set of end-to-end tasks, a reservation-based scheduling policy is needed to isolate the timing behavior of the components in run-time. This separation of run-time behavior for components is not supported by the above frameworks. Besides, we explicitly consider network resources in our framework, while the above frameworks only focus on the processor resources.

III. Model

We assume a Distributed Resource (DR) infrastructure with M resources. We use r_h to denote the h^{th} resource in the system. The set of resources is denoted by $\mathcal{R} = \{r_1, \dots, r_M\}$. We consider two types of resources: (i) network resources; (ii) processor resources. We assume that N real-time components are placed on the DR infrastructure. Each component uses a subset of all resources.

Component and task model. We assume that the $(\iota)^{th}$ real-time component $\mathcal{C}^{(\iota)}$ is composed of a set of tasks: $\mathcal{C}^{(\iota)} = \{\tau_1^{(\iota)}, \tau_2^{(\iota)}, \dots\}$, where $\tau_i^{(\iota)}$ represent the i^{th} task of $\mathcal{C}^{(\iota)}$. We assume an end-to-end sporadic task model in which a task $\tau_i^{(\iota)}$ requires a subset of available resources (processor and/or network) for completing its execution. $\tau_i^{(\iota)}$ begins its execution on a processor resource (source processor), and it finishes its execution on a processor resource (destination processor). The set of all resources consumed by $\tau_i^{(\iota)}$ is denoted using $\mathcal{R}_i^{(\iota)}$, where $\mathcal{R}_i^{(\iota)} \subset \mathcal{R}$. $\tau_i^{(\iota)}$ is characterized with a minimum inter-arrival time $p_i^{(\iota)}$ and an end-to-end soft deadline $d_i^{(\iota)}$. $p_i^{(\iota)}$ refers to the release of the task on the source processor, while $d_i^{(\iota)}$ indicates its relative deadline on the destination processor. $\tau_i^{(\iota)}$ is composed of a set of subtasks each consuming a resource. We use $\tau_{i,j}^{(\iota)}$ to refer to the j^{th} subtask of $\tau_i^{(\iota)}$. Note that we also use the term subtask for the chunks of the end-to-end tasks that consume the network resources (network subtasks). In this model, a message is a set of network subtasks. $\tau_{i,j}^{(\iota)}$ is characterized with a Resource Consumption Time (RCT) $c_{i,j}^{(\iota)}$ which indicates execution/transmission time of the subtask. We assume that the RCTs (i) are not known a priori to run-time; (ii) are changing during run-time. The quality of service experienced by tasks depends on the number of deadline violations. The objective of our adaptive framework is to minimize the number of deadline violations without significant resource overprovisioning. To this end, we use a controller module in our framework to track the resource requirements of the components and adjust the reservation budgets accordingly.

Virtual DR. Recall that each component is assigned to a subset of available resources denoted by $\mathcal{R}^{(\iota)}$. We use resource reservation polices for partitioning the bandwidth of the resources. For all $r_h \in \mathcal{R}^{(\iota)}$, $\mathcal{C}^{(\iota)}$ receives a fraction of the bandwidth of r_h. We refer to the subset of partially available resources for a component as a *virtual DR*, and we use $\Gamma^{(\iota)}$ for denoting it. The specification of the virtual DR allocated to $\mathcal{C}^{(\iota)}$ is denoted using:

$$\Gamma^{(\iota)} = \left\langle \Pi^{(\iota)}, \{\Theta_1^{(\iota)}, \Theta_2^{(\iota)}, \dots, \Theta_{M^{(\iota)}}^{(\iota)}\} \right\rangle,$$

where $\Pi^{(\iota)}$ is the period of the resource reservations, $\Theta_h^{(\iota)}$ denotes the budget of r_h reserved for $\mathcal{C}^{(\iota)}$ and $M^{(\iota)}$ is the number of resources used by $\mathcal{C}^{(\iota)}$. Without loss of generality, to avoid a conflict in resource indexing, we consider $M^{(\iota)} = M$. The above abstraction means that $\mathcal{C}^{(\iota)}$ is guaranteed to receive at least $\Theta_h^{(\iota)}$ time units of resource r_h every $\Pi^{(\iota)}$. The reserved bandwidth of $\Gamma^{(\iota)}$ on resource r_h is denoted by $\alpha_h^{(\iota)}$, and it is defined as:

$$\alpha_h^{(\iota)} = \frac{\Theta_h^{(\iota)}}{\Pi^{(\iota)}}.$$

Note that such a periodic resource abstraction model is supported by several processor and network scheduling schemes. For instance, on the processor resource we can use Constant Bandwidth Servers (CBS) [6] or Periodic Servers (PS) [1]. While for the network resource we can use a hierarchical server model [15] or the periodic model presented for the FTT-SE [13] and HaRTES [27] architectures.

IV. Framework

In this section we present the scheduling scheme of our framework. We also provide an overview of our adaptation mechanism.

Resource scheduler. We assume a resource scheduler per each physical resource. The schedulers (i) enforce resource reservations; (ii) schedule subtasks; (iii) and communicate with the respective controller modules to inform them about the state of the reservations. Although resources are scheduled locally, our end-to-end reservation scheme guarantees a predictable system wide resource provisioning for the end-to-end tasks. We use a hierarchical scheduling scheme in which scheduling is performed at two levels. In the higher level the resource scheduler schedules the reservations. Within each reservation, however, it is the responsibility of the subtask-scheduler to schedule different subtasks belonging to that reservation.

Controller module. We use a *controller* module per component which is responsible to adjust the reserved bandwidths for the component. The controller monitors the actual resource usages of the component on its allocated resources. This monitoring is performed through communications with the local schedulers. Once the controller decides a new set of reservation bandwidths, it communicates the new requirements to the local resource schedulers. The controller samples and adapts the system periodically. The sampling time is denoted by k. The time distance between two consecutive samples is referred as a sampling interval. We place the controller on a processor resource, and we reserve a portion of that processor resource for the controller executions as well as on the network resources that support the communication with the resource schedulers. These control reservations are static and attached to each component.

Since the reservation sizes are adapted during run-time, resources may temporarily become overloaded. In other words, the overall reserved bandwidth on a resource may pass beyond its schedulability threshold. In such a situation, the components can be prioritized based on their importance in their contribution to the overall goal of the system. If a component's bandwidth gets compressed on one of its resources, then it may be efficient to compress its bandwidth on the remaining component resources as well. Therefore, the framework should follow a protocol in overload situations. The overload management mechanism is out of the scope of this paper, and we leave incorporating an overload manager module to our framework for the future work. In this paper we intend to answer the following question:

> Given a component with requirements that vary dynamically on different resources, how we can define dynamic reservations that track the evolving requirements while satisfying resource constraints?

Example. In the following we present an example for elaborating our framework. In our example we assume a distributed system consisting of six resources $\{r_1, \ldots, r_6\}$, depicted in Figure 1. r_1, r_4 and r_5 are processor resources while r_2, r_3 and r_6 are network resources. We assume that a surveillance component has been placed on this distributed system. Two cameras are attached to r_1 and r_5. We have two end-to-end tasks: $\tau_1^{(1)}$ and $\tau_2^{(1)}$. The video frames are preprocessed in their source processors by the first subtasks of the two tasks. Thereafter, the video frames are sent to r_4 which hosts the final subtasks. We model this surveillance system as a component placed on the distributed infrastructure:

$$\mathcal{C}^{(1)} = \left\{\tau_1^{(1)}, \tau_2^{(1)}\right\}.$$

$\tau_1^{(1)}$ is composed of four subtasks: $\tau_{1,1}^{(1)}$,$\tau_{1,2}^{(1)}$,$\tau_{1,3}^{(1)}$ and $\tau_{1,4}^{(1)}$. $\tau_{1,1}^{(1)}$ represents the video encoder subtask placed on r_1, while $\tau_{1,4}^{(1)}$ is the decoder and display subtask on r_4. $\tau_{1,2}^{(1)}$ and $\tau_{1,3}^{(1)}$ represent the message that consumes r_2 and r_3 on its path from r_1 to r_4. Similarly $\tau_2^{(1)}$ is composed of four subtasks: $\tau_{2,1}^{(1)}$,$\tau_{2,2}^{(1)}$,$\tau_{2,3}^{(1)}$ and $\tau_{2,4}^{(1)}$ consuming r_5, r_6, r_3 and r_4. The resource reservations of $\mathcal{C}^{(1)}$ is represented using the following interface:

$$\Gamma^{(1)} = \left\langle \Pi^{(1)}, \{\Theta_1^{(1)}, \Theta_2^{(1)}, \Theta_3^{(1)}, \Theta_4^{(1)}, \Theta_5^{(1)}, \Theta_6^{(1)}\} \right\rangle.$$

Note that $\mathcal{C}^{(1)}$ may be sharing the resources with other components.

V. Component controller module

The objective of our framework is to satisfy the quality of service requirements of the tasks within the components. We consider meeting the end-to-end deadlines as a measure of the quality of service satisfaction. Our secondary objective is to allocate the resources efficiently. That is, the deadlines should be respected without significant resource overallocations. To this end, we design a feedback based controller module in this section.

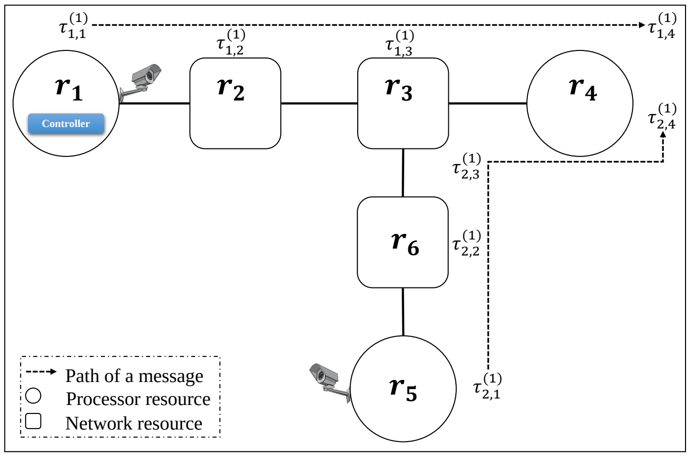

Fig. 1: Surveillance component example. r_1 and r_5 have two cameras attached. The lines connecting the resources represent logical connections. The network resources are explicitly visualized as boxes.

We use a control theoretic approach, similar to [28], for designing the component controller module. In such an approach, to control the plant, we define controlled variables, i.e., measurable variables that indicate the state of the plant, and control inputs, i.e., variables that allow us to manipulate the plant. Note that our plant is the set of resources that are used by the component. The component controller samples and adapts the plant periodically. In our framework, we consider one component controller per component. In the rest of this section, for notational convenience, we drop the component index (ι) when referring to the parameters associated with $\mathcal{C}^{(\iota)}$.

Let $\alpha_h(k)$ indicate the reserved bandwidth on r_h during sampling point $k-1$ and k. The component utilizes a portion of the reserved bandwidth. Let $\alpha'_h(k)$ denote the amount of consumed bandwidth during sampling point $k-1$ and k, where $0 \leq \alpha'(k) \leq \alpha(k)$. The amount of wasted bandwidth on r_h is: $y_h(k) = \alpha_h(k) - \alpha'_h(k)$, where we consider $y_h(k)$ the h^{th} output of our control system. Hence, the vector of system outputs is:

$$\mathbf{y}(k) = [y_1(k) \ldots y_M(k)]^T.$$

We selected the assigned bandwidth as our control inputs: $u_h(k) = \alpha_h(k) - \bar{\alpha}_h$, where $\bar{\alpha}_h$ is the operating bandwidth of the component on r_h. This parameter is provided by component developers. For instance it can be an estimate of the average required bandwidth on r_h. Note that this parameter does not need to be exact since we use a feedback loop to adjust the bandwidths. The vector of the control inputs is defined as follows:

$$\mathbf{u}(k) = [u_1(k) \ldots u_M(k)]^T.$$

At each sampling time, the goal of the component controller is to find a control input vector $\mathbf{u}(k)$ such that $\mathbf{y}(k) = \mathbf{y}^{ref}$, where $\mathbf{y}^{ref} = [y_1^{ref} \ldots y_{N^{(\iota)}}^{ref}]$, and y_h^{ref} is the desired value of $y_h(k)$. We assign this parameter to a small positive value to provide some slack bandwidth for the component. This is because $y_h(k)$ becomes saturated at zero, thus when $y_h(k) = 0$ it is not possible to infer whether the component requires more bandwidth or it is satisfied with the current bandwidth. To

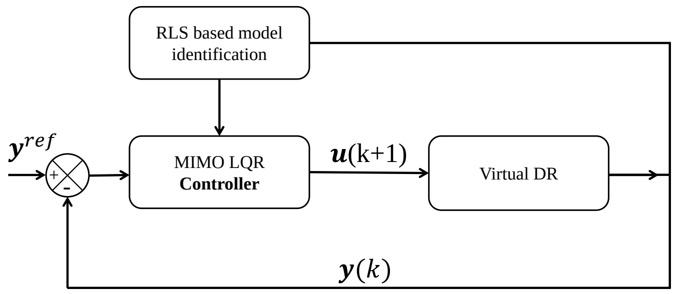

Fig. 2: The architecture of the control system.

reach the goal of the component controller, we use an on-line system identification method along with an optimal controller. Figure 2 illustrates the architecture of our control system. In the following we explain the details of the system identification method as well as the controller design. Note that, although we do not monitor the end-to-end response times, we can indirectly control the response times using our control inputs $\mathbf{u}(k)$. This is because by manipulating the bandwidth of a resource we can affect the response time of subtasks on that particular resource. Since we manipulate the bandwidth of all resources serving the end-to-end tasks, we can affect the end-to-end response times.

We need to model the relation between the control inputs $\mathbf{u}(k)$ and system output $\mathbf{y}(k)$. We use the following auto-regressive MIMO model for modeling this relation:

$$\mathbf{y}(k+1) = \mathbf{A}\mathbf{y}(k) + \mathbf{B}\mathbf{u}(k) + \varepsilon(k+1), \tag{1}$$

where $\mathbf{A}$ and $\mathbf{B}$ are $M \times M$ matrices, $\mathbf{u}(k)$ is the control input, $\mathbf{y}(k)$ is the system output, $\{\varepsilon(k+1)\}$ is a sequence of M-dimensional random vectors with zero mean representing the disturbance. $\mathbf{A}$ represents the dependency between the next wasted bandwidth and the current wasted bandwidth, whereas $\mathbf{B}$ represents the dependency between the current assigned bandwidth and the next wasted bandwidth. The use of a MIMO model allows us to capture the resource demand couplings among different resources of a component. We explain our approach for deriving $\mathbf{A}$ and $\mathbf{B}$ in the following subsection. Note that although the plant is non-linear by nature, it is known that linear models often work well for nonlinear systems [29].

A. System identification

In the above system model (Equation 1) matrices $\mathbf{A}$ and $\mathbf{B}$ are unknown. Since the load situation of the components may change during run-time, the above two matrices have to be tuned to improve the accuracy of the model. Therefore, we tune these matrices on-line at each sampling point. This self-learning technique enables the controller to learn the couplings among different system outputs/control inputs which may emerge during run-time. In this subsection we assume that $\mathbf{A}$ and $\mathbf{B}$ are also functions of the sampling time, i.e. $\mathbf{A}(k)$ and $\mathbf{B}(k)$. Let us rewrite the system model in the following form:

$$\mathbf{y}(k+1) = \mathbf{X}(k)\boldsymbol{\phi}(k) + \varepsilon(k+1), \tag{2}$$

where

$$\boldsymbol{\phi}(k) = \Big[(\mathbf{u})^T(k) \quad (\mathbf{y})^T(k)\Big]^T,$$
$$\mathbf{X}(k) = \Big[\mathbf{B}(k) \quad \mathbf{A}(k)\Big].$$

We identify $\mathbf{X}(k)$ during run-time by observing the system outputs, and by making a correction action. We use the Recursive Least Squares (RLS) method [30] for this purpose. In this method the estimated value of matrix $\mathbf{X}(k)$, denoted by $\hat{\mathbf{X}}(k)$, is calculated using the following equations:

$$\begin{gathered}\hat{\mathbf{X}}(k+1) = \hat{\mathbf{X}}(k) + \frac{\boldsymbol{\epsilon}(k+1)(\boldsymbol{\phi})^T(k)\mathbf{P}(k-1)}{\lambda + (\boldsymbol{\phi})^T(k)\mathbf{P}(k-1)\boldsymbol{\phi}(k)},\\ \boldsymbol{\epsilon}(k+1) = \mathbf{y}(k+1) - \hat{\mathbf{X}}(k)\boldsymbol{\phi}(k),\\ \mathbf{P}^{-1}(k) = \mathbf{P}^{-1}(k-1) + \Big(1 + (\lambda-1)\frac{(\boldsymbol{\phi})^T(k)\mathbf{P}(k-1)\boldsymbol{\phi}(k)}{[(\boldsymbol{\phi})^T(k)\boldsymbol{\phi}(k)]^2}\Big)\\ \boldsymbol{\phi}(k)(\boldsymbol{\phi})^T(k),\end{gathered} \tag{3}$$

where $\boldsymbol{\epsilon}(k+1)$ is the estimation error vector, $\mathbf{P}(k)$ is the covariance vector and λ is the forgetting factor [30].

B. Controller design

In the following, given that the system model is identified, we design an LQR controller which provides optimal control actions ($\mathbf{u}^*(k)$) at each sampling point k. The LQR controller design approaches allow component integrators to trade-off the speed of reaction against error sensitivity. This approach also allows the component integrators to assign different cost values on different resources, perhaps based on the expected demand fluctuations. Note that the LQR controller works after the RLS system identifier. Therefore, in this subsection we assume that the model parameters are already estimated, and we use $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ to indicate the estimated values of $\mathbf{A}$ and $\mathbf{B}$. We define error $\mathbf{e_P}(k)$ as:

$$\mathbf{e_P}(k) = \mathbf{y}^{ref} - \mathbf{y}(k). \tag{4}$$

The dynamics of the control system based on $\mathbf{e_P}(k)$ is as follows:

$$\begin{aligned}\mathbf{e_P}(k+1) &= \mathbf{y}^{ref} - \hat{\mathbf{A}}\mathbf{y}(k) - \hat{\mathbf{B}}\mathbf{u}(k)\\ &= \hat{\mathbf{A}}\mathbf{e_P}(k) - \hat{\mathbf{B}}\mathbf{u}(k) + (\mathbf{I} - \hat{\mathbf{A}})\mathbf{y}^{ref} - \varepsilon(k+1).\end{aligned}$$

Instead of directly using the model presented in Equation 1, we use the system model based on error for the controller design. In addition to $\mathbf{e_P}(k)$, we also use integral errors:

$$\mathbf{e_I}(k+1) = \mathbf{e_I}(k) + \mathbf{e_P}(k),$$

where $\forall k \leq 0$ we have $\mathbf{e_I}(k) = 0$. Hence, the augmented system model is:

$$\mathbf{e}(k+1) = \hat{\mathbf{H}}\mathbf{e}(k) + \hat{\mathbf{S}}\mathbf{u}(k) + \hat{\mathbf{L}} - \varepsilon(k+1), \tag{5}$$

where

$$\mathbf{e}(k) = \begin{bmatrix}\mathbf{e_P}(k)\\ \mathbf{e_I}(k)\end{bmatrix}, \quad \hat{\mathbf{H}} = \begin{bmatrix}\hat{\mathbf{A}} & \mathbf{0}\\ \mathbf{I} & \mathbf{I}\end{bmatrix},$$
$$\hat{\mathbf{S}} = \begin{bmatrix}-\hat{\mathbf{B}}\\ \mathbf{0}\end{bmatrix}, \quad \hat{\mathbf{L}} = \begin{bmatrix}\mathbf{I} - \hat{\mathbf{A}}\\ \mathbf{0}\end{bmatrix}\mathbf{y}^{ref}.$$

For our notational convenience we assume $\hat{\mathbf{\Psi}} = [\hat{\mathbf{S}} \quad \hat{\mathbf{H}}]$, and we rewrite the above equation:

$$\mathbf{e}(k+1) = \hat{\mathbf{\Psi}}\big[\mathbf{u}(k)^T \quad \mathbf{e}(k)^T\big]^T + \hat{\mathbf{L}} - \boldsymbol{\varepsilon}(k+1). \tag{6}$$

We define the objective function as:

$$J = E\Big\{\Big|\Big|\mathbf{W}(\mathbf{e}(k+1))\Big|\Big|^2 + \Big|\Big|\mathbf{Q}(\mathbf{u}(k) - \mathbf{u}(k-1))\Big|\Big|^2\Big\}, \tag{7}$$

where $\mathbf{W}$ and $\mathbf{Q}$ represent the cost of control error and the cost of control action, respectively. These two matrices allow the system designers to prioritize among different resources by assigning larger cost values to more important resources in case such a logical prioritization is needed. We derive the optimal control action by explicitly capturing the dependency of J on $\mathbf{u}(k)$, and by assigning the derivative of J with respect to $\mathbf{u}(k)$ equal to zero [28]. The optimal control action is:

$$\begin{aligned}\mathbf{u}^*(k+1) = &\Big((\mathbf{W}\hat{\mathbf{S}})^T\mathbf{W}\hat{\mathbf{S}} + \mathbf{Q}^T\mathbf{Q}\Big)^{-1}\Big[(\mathbf{W}\hat{\mathbf{S}})^T\mathbf{W} \\ &(\mathbf{L} - \hat{\mathbf{\Psi}}\tilde{\boldsymbol{\phi}}(k+1)) + \mathbf{Q}^T\mathbf{Q}(\mathbf{u})^T(k)\Big],\end{aligned} \tag{8}$$

where

$$\tilde{\boldsymbol{\phi}}(k+1) = [\mathbf{0}, (\mathbf{e})^T(k)]^T.$$

At each sampling time k the controller module takes the following actions:

1) It reads the system output vector $\mathbf{y}(k)$ provided by the local resource schedulers.
2) It updates its model of the plant's dynamics using Equation 3. The result system model is used in the next step.
3) It calculates the new control input vector $\mathbf{u}^*(k)$ using Equation 8.
4) It sends the new reservation bandwidths to the local resource schedulers.

The fact that we selected the wasted resource bandwidth as our system output ($\mathbf{y}$) offers the following advantages: (i) the adaptation scheme is independent of the number of tasks within the components; (ii) the adaptation scheme is independent of the assumed task model. In other words, the above adaptation scheme works under other task models, e.g. models in which a task can branch within its path from the source to different destinations. Nevertheless, in our evaluations we have only considered the task model presented in Section III.

VI. Evaluations

In order to perform the evaluations we have used a simulation tool, that is called SEtSim [31]. SEtSim was initially developed to support different real-time network protocols, such as the HaRTES architecture [27]. It was also recently extended to support AVB networks [32]. In this work, we modified the tool such that it supports our end-to-end task model as well as our multi-resource end-to-end reservation scheme. We consider the surveillance component case study presented in Figure 1. This is representative of typical network-based surveillance systems in which the bandwidth taken by each camera varies according to the scenario being captured, e.g., variable people walking, variable number of cars or other vehicles, and also where the cameras can be switched on and off on-line. We used a specific network technology for scheduling the network resource as well as a server-based scheduling policy for scheduling the processor resources. Both reservation mechanisms are compliant with the periodic abstraction model assumed in this paper (see section III). We present two simulations in this section. In the first simulation, we studied the response of our controller module to a step load change. That is, we used fixed RCTs times until a certain point in time. Thereafter, we increased the RCTs to larger numbers. In the second simulation, on the other hand, we used RCTs gathered from a real multimedia application to evaluate the performance of the controller module in a real scenario.

A. Simulation setup

In the following we explain the details of the resource reservation techniques as well as other parameters that are the same for the two case studies.

We used CBS [6] with hard replenishments on the processor resources. The hard CBS scheme works as follows. The server budget is periodically replenished to its maximum at each server release ($\Pi^{(\iota)}$). The tasks within the server are only allowed to run if the remaining budget is positive. The server budget is reduced while an active task belonging to the server is consuming the processor time. The tasks must stop as soon as the server budget is depleted.

The task parameters were set as follows. The periods of the two end-to-end task (p_1 and p_2) were set to $40ms$. Therefore, their first subtasks, i.e. $\tau_{1,1}$ and $\tau_{2,1}$, were released periodically with the same period. We assumed that $d_1 = d_2 = 40ms$. The messages are activated at the end of the execution of the sender subtasks, i.e. $\tau_{1,1}$ and $\tau_{2,1}$. The final subtasks, i.e. $\tau_{1,4}$ and $\tau_{2,4}$, were activated when they received their corresponding messages. We also set the reservation replenishment period (Π) to $40ms$. We used different RCTs for the two case studies.

We used the HaRTES architecture [27] to connect the processor nodes. Although we used a specific network technology in this evaluation, other network technologies that provide a resource management mechanism can be used in our framework. The HaRTES architecture uses modified Ethernet switches, so called HaRTES switches, which separate traffic in two classes, synchronous and asynchronous (Figure 3). The former is scheduled by the switch while the latter is shaped by the switch. Both the synchronous scheduling and the traffic shaping use the temporal resolution dictated by a pre-configured Elementary Cycle (EC). Within each EC, the traffic of each kind is confined to respective windows that vary dynamically in each network link according to the traffic needs (Figure 3).

In the case study, the messages are activated by the first subtasks, i.e. $\tau_{1,1}$ and $\tau_{2,1}$. Depending on the RCTs of the subtasks, the activation of the messages can be different. Therefore, we considered the messages as asynchronous traffic. A network perspective of the case study is shown in Figure 4. The network parameters were set as follows. The network bandwidth capacity was set to 100Mbps. The EC size was selected to be $40ms$. The bandwidth reservations for the transmission of the messages ($\alpha_h(k)$) were done within the asynchronous window and were changed by the controller during run-time depending on the load and bandwidth usage

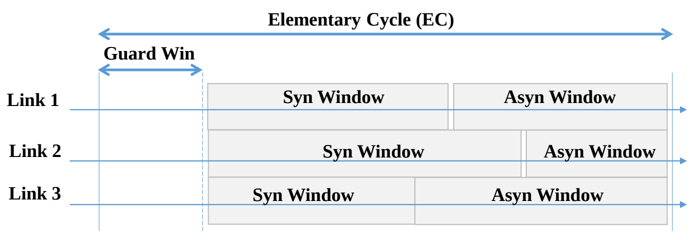

Fig. 3: The EC partitioning in the HaRTES architecture.

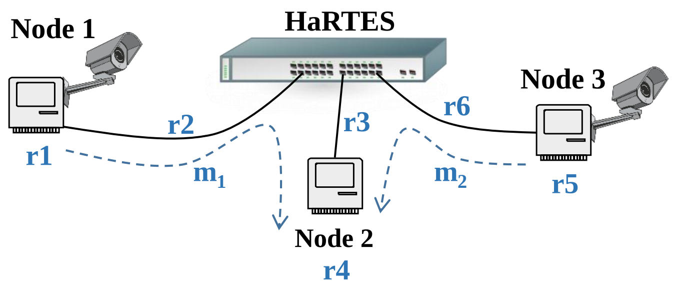

Fig. 4: The architecture of the case study from a network perspective. Figure 1 shows the resource perspective of the same system. Message one (m_1) denotes the following set of subtasks $\{\tau_{1,2}, \tau_{1,3}\}$, while message 2 ($m_2$) represents $\{\tau_{2,2}, \tau_{2,3}\}$.

	Before step	After step
$c_{1,1}$, $c_{1,4}$, $c_{2,1}$, $c_{2,4}$	$4ms$	$10ms$
$c_{1,2}$, $c_{2,2}$	$2ms$	$5ms$
$c_{1,3}$, $c_{2,3}$	$4ms$	$8ms$

TABLE I: The RCTs used in case study (1).

(resources r_2, r_3 and r_6). The processor nodes (Node 1 and Node 3 in Figure 4) generated two messages, respectively, denoted by m_1 and m_2 in the figure. The destination of the messages was Node 2, where the processing of the data was performed. We used fixed priority scheduling for scheduling subtasks within each resource reservation. We assumed that τ_1 (also its subtasks) had a higher priority than τ_2.

Regarding the controller parameters, we set the sampling interval to $200ms$. Also, we set $\mathbf{Q} = 0.1 \times \mathbf{I}$ and $\mathbf{W} = \text{tri}(0.1, 0.1, 0.1, 0.1, 0.1, 0.1) + \mathbf{I}$ to give a smaller weight for the integral errors. Note that tri returns a lower triangular matrix of its input vector. Based on the discussion presented in Section V, we assigned the following reference vector: $\mathbf{y}^{ref} = [0.08\ 0.10\ 0.15\ 0.08\ 0.08\ 0.10]^T$.

B. *Case study (1): step response*

Table I shows the RCTs used in this case study before and after a step change in the RCTs. We changed the RCTs $4s$ after the beginning of the simulations (i.e., after 100 task instances), and we ran the simulations for $10s$. The task response times are illustrated in Figure 5. During the transient period of load adjustment, a few instances of the end-to-end tasks missed their deadlines. τ_1 missed 20 deadlines ($\simeq 3.3$ %), while τ_2 missed 44 deadlines ($\simeq 7.3$ %). Recall that τ_2 had a lower priority than τ_1, hence it was scheduled later on the shared resources (still within the component reservations). In addition, since

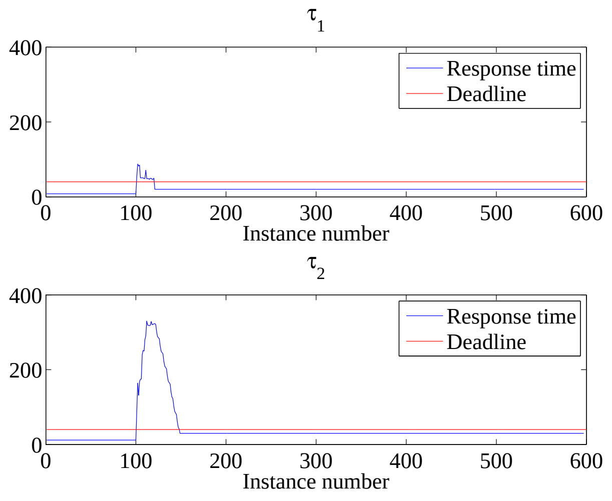

Fig. 5: End-to-end response times of τ_1 and τ_2 in case study (1).

after the step load a backlog was built, it took a while until the response times became stable again.

Figure 6 shows the consumed bandwidth α'_h, assigned bandwidth α_h, and the control error (Equation 4) for each resource separately. Although the RTCs were fixed throughout the first 10 samples, the controller modified the assigned bandwidths. This is because in the beginning of the simulations the controller needs to adjust its model. Let $\bar{\boldsymbol{\epsilon}}$ denote the average observed value of $\boldsymbol{\epsilon}(k)$ (Equation 3) which shows the accuracy of the system identification. We observed the following average differences between the estimated system outputs and the real system outputs:

$$\bar{\boldsymbol{\epsilon}} = 10^{-4} \times [-74;\ 49;\ 54;\ -107;\ -75;\ 48],$$

which shows that the RLS approach has been successful in identifying the system model.

C. *Case study (2): multimedia application*

In this simulation, in order to have realistic evaluation of our framework, we used RCTs gathered from running multimedia tasks on a real hardware platform [33]. We also considered 10 % high priority interfering workload on all of the resources to simulate the existence of other components. Figure 7 presents the evolution of the used bandwidths, assigned bandwidths and control errors on all resources during the $100s$ experiment (i.e., 500 control samples). The figure shows that the controller managed to successfully track the evolution of the workload. Figure 8 illustrates the response times of the two end-to-end tasks. Since τ_2 had a lower priority than τ_1, it had larger response times, and it occasionally violated its deadline. In total, τ_1 missed 4 deadlines ($\simeq 0.1$ %), while τ_2 missed 40 deadlines ($\simeq 1.6$ %). We observed:

$$\bar{\boldsymbol{\epsilon}} = 10^{-4} \times [5;\ 1;\ -9;\ 15;\ 2;\ 16].$$

Let $\boldsymbol{\alpha}^{avg}$ denote the vector of average assigned bandwidths throughout the experiment. We had:

$$\boldsymbol{\alpha}^{avg} = [0.1977;\ 0.2600;\ 0.6378;\ 0.3686;\ 0.2030;\ 0.2174],$$

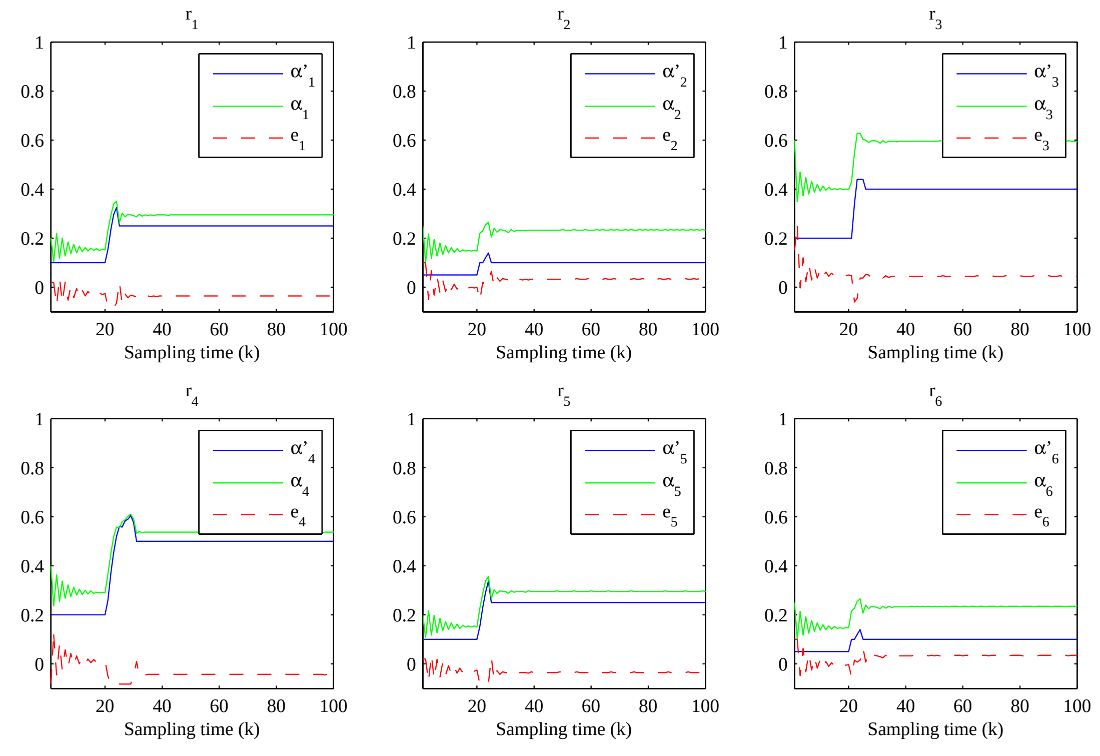

Fig. 6: Consumed bandwidth α'_h, assigned bandwidth α_h and control error e_h of the six resources during case study (1).

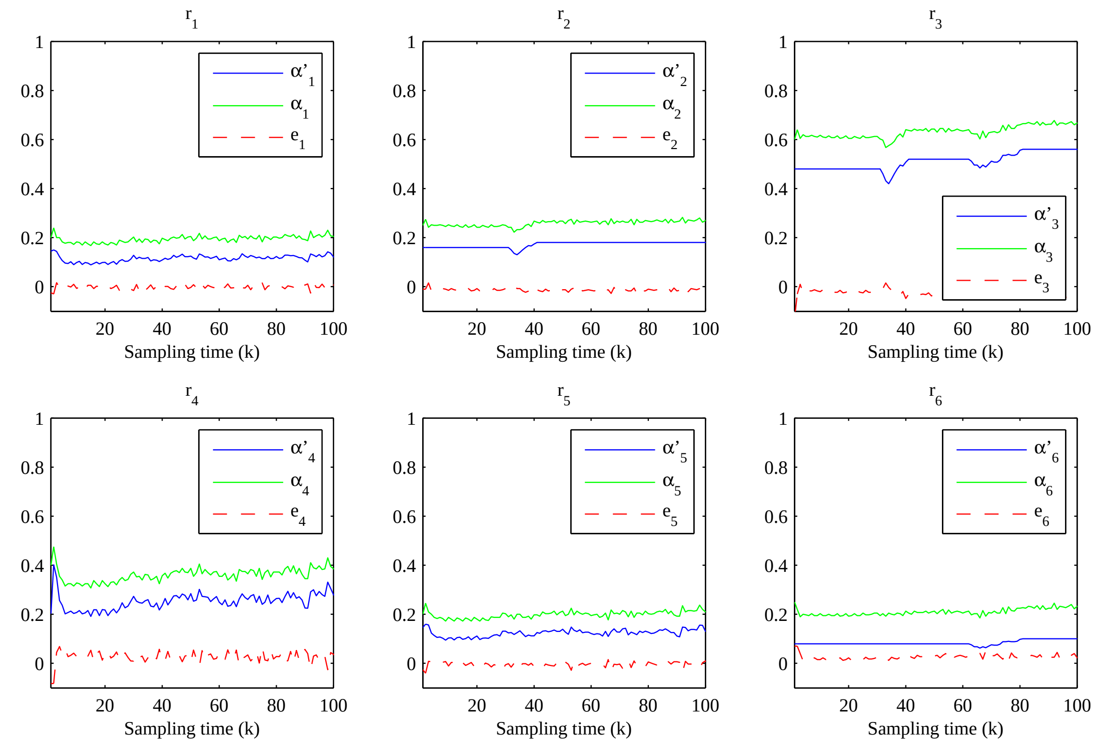

Fig. 7: Consumed bandwidth α'_h, assigned bandwidth α_h and control error e_h of the six resources during case study (2).

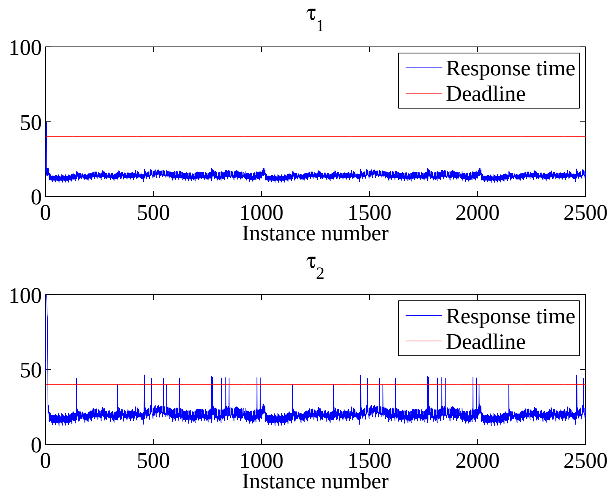

Fig. 8: End-to-end response times of τ_1 and τ_2 in case study (2).

which shows the high accuracy of the system identification.

In a new experiment, we used the above observed average bandwidths $\boldsymbol{\alpha}^{avg}$, and we assigned them as fixed bandwidths to the component. In total, the number of deadline misses for τ_2 was 46 ($\simeq 1.8$ %), while τ_1 missed all of its deadlines. This is because the first instance of $\tau_{1,1}$ finished in the 6^{th} reservation period. This phenomenon caused a large backlog for τ_1. On the other hand, since α_2 was not enough for sending multiple messages, τ_1 never recovered from this backlog. The average response time for τ_1 was $248ms$. This experiment shows that given a certain resource efficiency level (i.e. reservation bandwidths) our adaptive framework works significantly better than the static design approach. It should be noted that the static design approach (i.e., the above static bandwidth assignments) requires bandwidth estimations prior to the run-time, while our adaptive approach does not have such a requirement.

D. Overhead

Our adaptive framework performs adaptations at the cost of imposing two types of overhead: (i) communication overhead; (ii) computation overhead. The controller requires the bandwidth usage information ($\mathbf{y}(k)$) during run-time to adjust the bandwidth for each network resource. This information is gathered by the resource scheduler in the nodes and switches per link, and it is transmitted to the controller by means of messages. Therefore, besides the data messages transmitted through the network, the controller messages are sent through the same links. For this particular information a specific bandwidth is reserved on the network, that is isolated from the data message bandwidth. For instance, in the HaRTES architecture, the data messages are transmitted within the asynchronous window, while the control messages are transmitted within the synchronous window. Note that the controller is performed periodically, thus the control messages can be transmitted periodically as a set of synchronous messages. A small portion of the synchronous window can be reserved for the control messages. For the control purpose we can send messages with the size equal to 500 Bytes ($40\mu s$) every reservation period ($40ms$) which imposes 0.1 % overhead on the network. The number of control messages can be reduced by devising a decentralized control scheme similar to [34], where we can decompose the system model presented in Equation 1. However, we leave investigating this approach for the future work.

Furthermore, the controller performs computations to first update its model of the system using the RLS technique (Equation 3), and to calculate the control input $\mathbf{u}(k)$ (Equation 8). For the purpose of the above case study, we implemented the two functions in Matlab, and we ran them on our machine featuring an Intel Core i7-4600U processor with 12 GB RAM. The RLS identification took in average $373\mu s$, while the LQR computations took in average $240\mu s$. Therefore, given the sampling period, the controller module imposed around 0.31 % computation overhead. Note that the above values are measured running Matlab code on a Windows operating system. Therefore, an efficient implementation executed on a real-time operating system will potentially impose lower overhead. In general, overhead is proportional to the total number of resources used by the component $M^{(\iota)}$.

E. Discussions

We conclude from the above case studies that our controller module manages to fulfill the objectives described in Section V, that are (i) to serve the components with negligible deadline violations; (ii) to allocate the resources efficiently by matching the reservation bandwidths to the instantaneous needs during run-time (instead of overprovisioning the resources). Considering the network resources in the evaluation, we changed the reservations within the asynchronous window sizes in the EC during run-time. This allows other components in the asynchronous window to utilize more bandwidth. Also, we can reduce the size of the asynchronous window if it is not used by the components within this window. In doing so, the size of the other window, i.e., the synchronous window can be increased. Therefore, more resource is available for the components that use the synchronous window in the EC. The same argument applies for the processor resources. The component allows other components that share the resources with itself to utilize larger bandwidths by keeping its reservation size low.

VII. Conclusions and Future Work

In this paper we designed an adaptive framework for scheduling component-based distributed real-time systems. In our framework we enforce end-to-end reservations across all of the resources needed by the end-to-end tasks within the components. The sizes of the reservations are adjusted during run-time to cope with dynamic resource needs. We showed, using two case studies, that our framework reduces the number of deadline violations to a negligible level, while keeping the reservation sizes close the actual demands.

In the future we will propose a protocol to manage overload situations in which the overall resource reservation on a resource is beyond its schedulability threshold. In addition, we would like to investigate using a decentralized control approach to see whether we can reduce the communication overhead of the controller while keeping its performance at an acceptable level.

REFERENCES

[1] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *RTSS'03*, December 2003, pp. 2–13.

[2] G. Lipari and S. Baruah, "A hierarchical extension to the constant bandwidth server framework," in *RTAS'01*, May 2001, pp. 26–35.

[3] Z. Deng and J. W.-S. Liu, "Scheduling real-time applications in an open environment," in *RTSS'97*, December 1997, pp. 308–319.

[4] L. Palopoli, T. Cucinotta, L. Marzario, and G. Lipari, "AQuoSA-adaptive quality of service architecture," *Software: Practice and Experience*, vol. 39, no. 1, pp. 1–31, January 2009.

[5] N. Khalilzad, M. Behnam, and T. Nolte, "Multi-level adaptive hierarchical scheduling framework for composing real-time systems," in *RTCSA'13*, August 2013, pp. 320–329.

[6] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *RTSS'98*, December 1998, pp. 4–13.

[7] D. Faggioli, M. Trimarchi, F. Checconi, M. Bertogna, and A. Mancina, "An implementation of the earliest deadline first algorithm in Linux," in *SAC'09*, March 2009, pp. 1984–1989.

[8] M. Behnam, T. Nolte, I. Shin, M. Åsberg, and R. J. Bril, "Towards hierarchical scheduling on top of VxWorks," in *OSPERT'08*, July 2008, pp. 63–72.

[9] E. Bini, G. Buttazzo, J. Eker, S. Schorr, R. Guerra, G. Fohler, K.-E. Årzen, V. Romero, and C. Scordino, "Resource management on multicore systems: The ACTORS approach," *Micro, IEEE*, vol. 31, no. 3, pp. 72–81, May-June 2011.

[10] J. Loeser and H. Haertig, "Low-latency hard real-time communication over switched ethernet," in *ECRTS'04*, June 2004.

[11] IEEE, "IEEE Std. 802.1qav, ieee standard for local and metropolitan area networks, virtual bridged local areanetworks, amendment 12: Forwarding and queuing enhancements for time-sensitive streams." IEEE, Tech. Rep., 2011.

[12] *EPSG Draft Standard 301 Ethernet POWERLINK Communication Profile Specification Version 1.2.0*, Ethernet POWERLINK Standardisation Group, 2013.

[13] M. Ashjaei, M. Behnam, L. Almeida, and T. Nolte, "Performance analysis of master-slave multi-hop switched ethernet networks," in *SIES'13*, June 2013.

[14] R. Santos, A. Vieira, P. Pedreiras, A. Oliveira, L. Almeida, R. Marau, and T. Nolte, "Flexible, efficient and robust real-time communication with server-based Ethernet switching," in *WFCS'10*, May 2010.

[15] R. Santos, M. Behnam, T. Nolte, P. Pedreiras, and L. Almeida, "Multi-level hierarchical scheduling in ethernet switches," in *EMSOFT'11*, October 2011.

[16] J. Silvestre-Blanes, L. Almeida, R. Marau, and P. Pedreiras, "Online QoS management for multimedia real-time transmission in industrial networks," *IEEE Transaction on Industrial Electronics*, vol. 58, no. 3, March 2011.

[17] "IEEE 802.1Qat, draft standard for local and metropolitan area networks virtual bridged local area networks amendment 9: Stream reservation protocol (SRP)."

[18] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: a new resource reservation protocol," *IEEE Communications Magazine*, vol. 40, no. 5, pp. 116–127, May 2002.

[19] K. Lakshmanan and R. Rajkumar, "Distributed resource kernels: Os support for end-to-end resource isolation," in *RTAS'08*, April 2008.

[20] A. Oliveira, A. Azim, S. Fischmeister, R. Marau, and L. Almeida, "D-RES: Correct transitive distributed service sharing," in *ETFA'14*, September 2014.

[21] T. Cucinotta and L. Palopoli, "QoS control for pipelines of tasks using multiple resources," *IEEE Transactions on Computers*, vol. 59, no. 3, pp. 416–430, March 2010.

[22] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, "A resource allocation model for QoS management," in *RTSS'97*, December 1997.

[23] S. Ghosh, J. Hansen, R. Rajkumar, and J. Lehoczky, "Integrated resource management and scheduling with multi-resource constraints," in *RTSS'04*, December 2004.

[24] J. Stankovic, T. He, T. Abdelzaher, M. Marley, G. Tao, S. Son, and C. Lu, "Feedback control scheduling in distributed real-time systems," in *RTSS'01*, December 2001, pp. 59–70.

[25] C. Lu, X. Wang, and X. Koutsoukos, "Feedback utilization control in distributed real-time systems with end-to-end tasks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 6, pp. 550–561, June 2005.

[26] X. Wang, D. Jia, C. Lu, and X. Koutsoukos, "DEUCON: Decentralized end-to-end utilization control for distributed real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 7, pp. 996–1009, July 2007.

[27] M. Ashjaei, M. Behnam, P. Pedreiras, R. J. Bril, L. Almeida, and T. Nolte, "Reduced buffering solution for multi-hop HaRTES switched Ethernet networks," in *RTCSA'14*, August 2014.

[28] X. Liu, X. Zhu, P. Padala, Z. Wang, and S. Singhal, "Optimal multivariate control for differentiated services on a shared hosting platform," in *CDC'07*, December 2007, pp. 3792–3799.

[29] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[30] K. J. Åström and B. Wittenmark, *Adaptive control*, 2nd ed. Reading, Mass.: Addison-Wesley, 1995.

[31] M. Ashjaei, M. Behnam, and T. Nolte, "The design and implementation of a simulator for switched ethernet networks," in *WATERS'12*, July 2012.

[32] ——, "SEtSim: A modular simulation tool for switched Ethernet networks," Tech. Rep., September 2014. [Online]. Available: http://www.es.mdh.se/publications/3706-

[33] C. C. Wust, L. Steffens, W. F. J. Verhaegh, R. J. Bril, and C. Hentschel, "QoS control strategies for high-quality video processing," *Real-Time Systems*, pp. 3–12, 2005.

[34] J. Yao, X. Liu, X. Chen, X. Wang, and J. Li, "Online decentralized adaptive optimal controller design of cpu utilization for distributed real-time embedded systems," in *ACC'10*, June 2010, pp. 283–288.