Multi-level Hierarchical Scheduling in Ethernet Switches

Rui Santos
IEETA / University of Aveiro
Aveiro, Portugal
rsantos@ua.pt

Moris Behnam
MRTC / Mälardalen University
Västerås, Sweden
moris.behnam@mdh.se

Thomas Nolte
MRTC / Mälardalen University
Västerås, Sweden
thomas.nolte@mdh.se

Paulo Pedreiras
IEETA / University of Aveiro
Aveiro, Portugal
pbrp@ua.pt

Luís Almeida
University of Porto
Porto, Portugal
lda@fe.up.pt

ABSTRACT

The complexity of Networked Embedded Systems (NES) has been growing steeply, due to increases both in size and functionality, and is becoming a major development concern. This situation is pushing for paradigm changes in NES design methodologies towards higher composability and flexibility. Component-oriented design technologies, in particular supported by server-based scheduling, seem to be good candidates to provide the needed properties.

As a response we developed a multi-level hierarchical server-based architecture for Ethernet switches that provides composability and supports online adaptation and reconfiguration. This paper extends our work, presenting the associated response-time based schedulability analysis, necessary for the admission control procedure. Additionally, we have derived the temporal complexity of the analysis, which is shown to be $O(n^2)$, where n is the number of higher priority components associated with a given server. Finally, we present a proof-of-concept implementation and a set of experimental results that validates the analysis.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Network communications, Distributed networks*

General Terms

Algorithms, Design, Performance

Keywords

Hierarchical Scheduling, Real-Time Systems, Real-Time Ethernet, Real-Time Communications, Response-Time Analysis

EMSOFT'11, October 9–14, 2011, Taipei, Taiwan.

1. INTRODUCTION

Networked Embedded Systems (NES) are becoming large scale distributed systems composed of networked nodes that increasingly embed more functionality and exchange higher amounts of and more heterogeneous data. This scenario is still aggravated by the growing difficulty in fully configuring the system at design time and the consequent need for run-time adaptation. Such adaptation is necessary to make better use of the computational and communications resources, taking actual operational environment and state of systems components into consideration.

One suitable methodology to tackle such increasing complexity is component-oriented design. It enables composability and can be effectively deployed using server-based architectures that are an effective means to enable controlled resource sharing. In particular, these techniques allow transparently providing the applications with virtual resources that are a fraction of the capacity of the corresponding hardware resources, while offering mutual temporal isolation. Thus, servers can provide composability among different applications and, when organized hierarchically, they can also provide composability among different applications components. In this paper we use hierarchical server scheduling to enable composability among streams of messages in NES. Enabling such composability allows us to deal with complex systems in a flexible manner.

Frequently NES are used in application domains that have intrinsic real-time requirements, e.g., in distributed real-time control systems that closely interact with the physical environment. In such cases special-purpose real-time communication networks are used to achieve the required timeliness properties. Real-Time Ethernet-based protocols (RTE), such as PROFINET, EtherCAT, Ethernet POWERLINK and TTEthernet, are emerging as the de facto communication technologies for NES, taking advantage of the appealing attributes of the underlying Ethernet technology. However, currently available RTE protocols do not employ efficient server-based scheduling policies, e.g., policies as those developed for CPU scheduling. Even when available, network partitions are typically static, as in TDMA-based approaches, and do not adapt to variations in number of active components in the system or in their requirements. Moreover, the respect for network partitions is frequently delegated to the end nodes that must execute a specific layer on top of the general network interface, typically a traffic shaper, which is a limitation for the integration of

legacy systems and other general purpose systems that do not originally include such layer. Finally, even in the cases in which such layer can be effectively deployed, the proper operation of the system requires the compliance of all system components to a achieve a correct temporal behavior.

These observations motivate the development of a novel multi-level hierarchical server-based architecture for Ethernet switches. This architecture allows for a hierarchical composition of servers that enable division of the network bandwidth in a hierarchical way, creating virtual channels with temporal isolation among them, thus supporting composability in the time domain. Due to the multi-level hierarchy, this composability applies not only to different applications that might coexist in the same node but also to different streams of the same applications, e.g., data and control streams in multimedia streaming.

Moreover, to efficiently cope with applications that exhibit evolving requirements, the hierarchical server framework supports both dynamic server adaptation, i.e., the server attributes such as capacity and period may be adapted online, and hierarchy reconfiguration, i.e., servers may be added, moved to different branches or even removed. The adaptation and reconfiguration services are accessed via reconfiguration request messages issued by the nodes to the switch.

NES are often used in critical applications, and thus temporal requirements behavior must be guaranteed by design. To achieve such level of guarantees, we force all change requests through an admission control procedure that performs a schedulability analysis test and verifies if the Ethernet switch has enough resources to accommodate the resulting configuration (e.g. memory, server structures). The automotive domain is an example where the flexibility (adaptation and reconfiguration) is important to integrate components progressively without needing to reanalyze the whole system, but just with incremental analysis.

This paper presents the following contributions:

- a new architecture for Ethernet switches that provides a dynamically reconfigurable and adaptable multi-level hierarchy of temporally isolated virtual channels;
- a new response-time based schedulability analysis for multi-level hierarchical scheduling on switched Ethernet networks, suitable to the proposed architecture;
- the analysis of the respective temporal complexity, which is shown to be $O(n^2)$, where n is the number of higher priority components associated with a given server;
- and a proof-of-concept implementation that validates the proposed architecture and the analysis with experimental data.

The remainder of the paper is organized as follows. Section 2 presents an overview of related work, and Section 3 outlines the system model. Section 4 presents the response time schedulability analysis while Section 5 presents the algorithm for computing the exact response time. Section 6 presents a prototype implementation of the framework and Section 7 presents experimental results that validate the analysis and respective implementation. Finally, Section 8 concludes the paper.

2. RELATED WORK

The use of servers in networking is common, being the *leaky bucket* the most well-known. The leaky-bucket is, in fact, part of a general server category called *traffic shapers* [9], which purpose is to limit the amount of traffic that a node can submit to the network within a given time window, bounding the node burstiness. These servers use techniques similar to those used by CPU servers, based on capacity that is eventually replenished. Many different replenishment policies are also possible, being the periodic replenishment as with the Polling Server (PS) or the Deferrable Server (DS), the most common ones.

Particularly regarding Real-Time Ethernet (RTE) protocols, some very limited forms of server-based traffic handling can also be found. Some protocols enforce periodic communication cycles with reserved windows for different traffic classes (e.g., PROFINET-IRT [11] and Ethernet POWERLINK [1]). This is a trivial composition of several PS that results in an inefficient use of the network bandwidth. Other protocols, such as [9], implement traffic shapers in the end nodes that behave similarly to a DS. However, infrastructural limitations, which can be solved by the framework presented in this paper, prevent these protocols from supporting arbitrary server policies or hierarchical composition and dynamic reconfiguration.

Recent QoS-enabled Ethernet switches, e.g., Cisco Catalyst WS-C3560-8PC-S, already include enhanced features such as advanced QoS control and rate limitation. The latter operates like a server, allowing to control the maximum bandwidth allocated to a user entity but fine control over the server temporal parameters, support for composition and admission control are still lacking. Enhanced architectures such as that proposed in [16] rely on clock-driven scheduling which approximates a PS and use static traffic definition to optimize the scheduling.

Another related area, despite typically considering preemptive task scheduling, is that of general hierarchical scheduling frameworks (HSF). Deng and Liu [6] began proposing two-level HSF for open systems, where subsystems may be developed and validated independently. Kuo and Li [8] introduce for such two-level HSF a schedulability analysis based on FPS with a global scheduler. Shin and Lee [15] present a generic scheduling interface model in order to construct hierarchical scheduling frameworks. Almeida and Pedreiras [2] present a response time analysis for the periodic server model and address the problem of designing a server to fulfill the application constraints. Arvind *et al.* [7] generalize the periodic resource model for compositional analysis of hierarchical scheduling frameworks. Finally, another related area is that of synchronization protocols in HSF. For example, SIRAP [3] addresses CPU resource sharing among several subsystems that execute within servers and it proposes inserting idle-time whenever the remaining capacity is not enough to execute an access to a shared resource. In the specific case of non-preemptive scheduling, which implies an exclusive access to a shared resource, SIRAP becomes similar to the techniques presented in [10] for the scheduling of the asynchronous traffic in FTT-CAN. Nevertheless, both cases address two-levels HSFs, only, while in this paper we seek explicitly the support to multi-level HSF.

Finally, another analytical techniques using the Network Calculus can also be considered, however they use a analysis model based on the burstiness and long-term rate, which is

different from the analysis proposed in this paper, base on the periodic model.

3. MODEL DEFINITION

This paper defines and formalizes a general multi-level hierarchical server-based scheduling framework that efficiently handles the distribution of shared resources. While the focus is on the distribution of bandwidth in switched Ethernet networks, the concept can be generalized and reused in other network technologies or even on other fields with similar requirements and constraints. The proposed framework allows dividing and subdividing the network bandwidth (resource) among different streams, in a hierarchical way, mirroring the composition of the systems, while providing real-time guarantees (bandwidth and response time) and temporal isolation.

This hierarchical structure can be represented as a tree (Figure 1). Each node of the hierarchy represents a server that handles a portion of bandwidth and, associated to this server (parent), several servers (children) can be connected that share the parent bandwidth. This procedure can be iterated as many times as required, resulting in a multi-level server hierarchy. A scheduler attached to each parent server regulates the access of the children servers to the bandwidth it provides. The streams, which are the entities that actually consume the bandwidth, are placed at the end of the branches, i.e., the leaves.

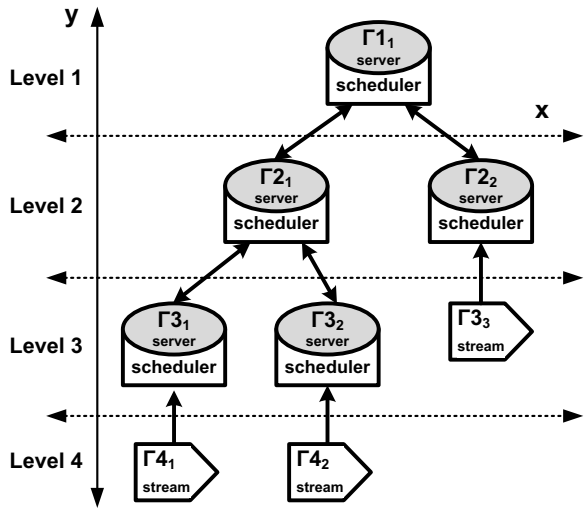

Figure 1: Server Hierarchy.

As mentioned in Section 1, the hierarchical server framework herein presented features dynamic adaptability and reconfigurability, i.e., it allows adding and removing servers and streams, as well as updating attributes at runtime without disrupting the switch operation. To this end, a management interface allows the nodes to communicate hierarchy update requests to the switch. To ensure a correct real-time behavior, all such requests are subject to an admission control based on the schedulability analysis algorithm presented in Section 5. This algorithm verifies if all server and stream deadlines can be met and only in such case the server hierarchy is updated.

The framework proposed in this paper presents some constraints that have relevant impact in the schedulability analysis. As usual in communication protocols, the preemption of packets that are in transmission is not allowed. Moreover, the server capacity is strictly enforced and thus overruns cannot occur. Consequently, idle time may appear at the end of each server instance whenever the capacity available is not enough to transmit the next packet.

3.1 Servers and streams

In the scope of this paper a component Γ_{y_x} is identified by indexes y and x, where y identifies the level in the hierarchy and x identifies the component inside that level (Figure 1). This way, $y = 1, .., NL$ and $x = 1, .., NC_y$, where NL is the maximum number of levels in the hierarchy and NC_y is the maximum number of components in the level y.

The asynchronous streams are at the end of the hierarchy and they are characterized in (1) using the sporadic real-time model, where C_{y_x} is the message transmission time of a stream instance, $Tmit_{y_x}$ represents the respective minimum interarrival time and D_{y_x} the deadline. It is assumed that a message stream instance may generate several packets, which have a size comprised between $Mmin_{y_x}$ and $Mmax_{y_x}$. P_{y_x} identifies the parent server, i.e, the server to which the stream is connected to and RT_{y_x} is its computed response time.

$$AS_{y_x} = (C_{y_x}, Tmit_{y_x}, Mmax_{y_x}, Mmin_{y_x}, P_{y_x}, RT_{y_x}, D_{y_x}) \quad (1)$$

A server Srv_{y_x} is characterized in (2) by its capacity C_{y_x}, replenishment period T_{y_x}, deadline D_{y_x} and a few data extracted from the set of children components, either servers or streams, namely the maximum and minimum packet transmission times ($Mmax_{y_x}$ and $Mmin_{y_x}$, respectively). Moreover, the server Srv_{y_x} is associated with a parent server P_{y_x} and a corresponding computed upper bound response time RT_{y_x}. Despite the similarity between the characterization of servers and streams, there is a fundamental difference since only streams imply actual transmission time that uses the capacity of the respective servers. Servers merely characterize a reservation of the network resource.

$$Srv_{y_x} = (C_{y_x}, T_{y_x}, Mmax_{y_x}, Mmin_{y_x}, P_{y_x}, RT_{y_x}, D_{y_x}) \quad (2)$$

In the remainder of the paper we will refer to both streams and servers as components, in an integrated way.

4. SECHEDULABILITY ANALYSIS

The schedulability analysis is performed by the admission controller to verify if eventual change requests to the server hierarchy are feasible, i.e., it checks if the change results in a configuration in which all the component deadlines are met. This section introduces a response-time schedulability analysis algorithm, based on deadline monotonic scheduling, for a generic hierarchical structure of servers and streams as presented above. However, it is equally applicable to arbitrary fixed priorities in which case the models in (1) and (2) must be extended with a priority.

With respect to previous related work of the authors [12] and [3], the analysis in this paper accounts for inserted idle-time in a different more efficient manner and considers the

preemption between packets, but not the preemptive transmission of packets. Both innovations result in increased accuracy as shown later on in the experiments.

4.1 Schedulability analysis algorithm

As referred before, server capacities are strictly enforced and overruns, e.g., caused by a non-preemptive packet transmission that extends beyond the exhaustion of the respective server capacity, are not allowed. This is avoided inserting idle-time, called *self-blocking* in the scope of SIRAP [3], whenever the remaining server capacity is not enough for the transmission of a full packet. This way, the remaining capacity is wasted and the pending transmission is delayed for successive server instances when enough capacity is available (Figure 2). Therefore, the maximum inserted idle-time that a server component Γ_{y_x} can suffer is equal to the maximum packet transmission time managed by this server ($Mmax_{y_x}$). This value also allows knowing which is the maximum blocking caused by the respective component Γ_{y_x} to the higher priority components in the same branch and in the same level. On the other hand, $Mmin_{y_x}$ is used when considering non-preemption in the computation of the higher priority interference and to compute the maximum memory required in each branch. This latter subject, however, is out of the scope of this paper.

When a change in the hierarchical structure is requested, for instance adding or removing a stream, it may have impact in $Mmax$ and $Mmin$ along the hierarchy. Therefore, the schedulability algorithm presented in this section is executed in two phases, as shown below.

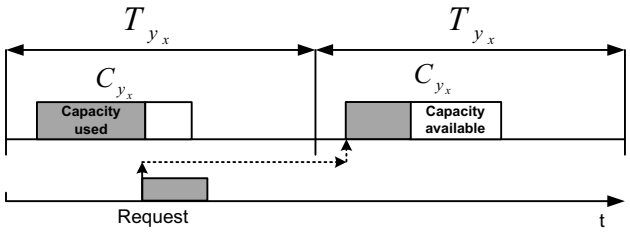

Figure 2: Inserting idle-time to enforce servers capacities.

4.1.1 Schedulability analysis algorithm - 1^{st} phase

This phase evaluates, for each sever in the hierarchy path, the impact in $Mmax$ and $Mmin$ of the change request. This procedure is carried out from the bottom to the top, propagating the maximum blocking time and memory requirements to the parent servers. At the end of this phase, the top level component Γ_{1_1} will have the maximum ($Mmax$) and minimum ($Mmin$) packet transmission times among all the streams transmitted. Correspondingly, each server Γ_{x_y} will inherit $Mmax$ and $Mmin$ from all its children. An alternative to this step would be to use the maximum packet size allowed by the technology. However, despite the simplification, it would generate unnecessary pessimism.

Moreover, this phase also verifies the necessary condition described in (3) to see if all servers have capacity at least as large as the maximum packet they need to handle. If this condition fails the analysis stops and the current configuration remains unchanged.

$$\forall \Gamma_{y_x, y=1..NL, x=1..NC_y}, C_{y_x} \geq Mmax_{y_x} \tag{3}$$

4.1.2 Schedulability analysis algorithm - 2^{nd} phase

This phase consists in computing the components worst-case response-times RT_{y_x} and comparing with the respective deadlines to assess the schedulability of the hierarchy.

To compute the worst-case response times we use the typical technique in hierarchical scheduling based on a request bound function **rbf** that models the load submitted to a server and a supply bound function **sbf** that models the capacity (bandwidth) provided by a server.

In this case, we define $\mathbf{rbf}_{y_x}(t)$ as the request bound function of the server Γ_{y_x} that quantifies the maximum load submitted up to instant t to the parent component P_{y_x} by the component itself together with the interference of higher priority components and the blocking of low priority components. Then we define $\mathbf{sbf}_{P_{y_x}}(t)$ as the supply bound function associated to the parent component of Γ_{y_x} that computes the minimum bandwidth supply provided to its children at instant t.

As referred, one important aspect that we are considering in this analysis, that was not considered in [12], is that preemption is allowed between packets, only, i.e., packets transmission is non-preemptive. Particularly, any higher priority components that become ready during the transmission of the last packet (of a stream or server instance) will not be able to preempt it and thus will not impact on the component response time. If full preemption was considered, more interference could potentially be accounted for and the worst-case response time estimate would yield more pessimistic.

However, the model does not explicitly say which is the length of the last packet (M^{last}). Thus, the safe choice is to set $M^{last} = Mmin$ which maximizes the interference that a component can suffer.

The response time RT_{y_x} is then computed as in (4), where w_{y_x} is the busy interval for the Γ_{y_x} component. This interval is the time lapse from when Γ_{y_x} becomes ready until it starts transmitting its last packet, considering all higher priority load that becomes ready in the meanwhile.

$$\begin{aligned} RT_{y_x} &= w_{y_x} + M^{last}_{y_x}, \\ w_{y_x} &= \text{ earliest } t > 0 : \mathbf{rbf}_{y_x}(t) = \mathbf{sbf}_{P_{y_x}}(t) \end{aligned} \tag{4}$$

where $M^{last}_{y_x} = Mmin_{y_x}$.

The request bound function $\mathbf{rbf}_{y_x}(t)$ is now computed as in (5) where $IH_{y_x}(t)$ is the higher priority load generated up to instant t and submitted to the same parent component $\Gamma_{P_{y_x}}$ as given by (6), and BL_{y_x} (7) is a blocking term associated to the non-preemptive nature of the packets transmission and maximized by the largest M_{max} among all lower or equal priority components $lpe(\Gamma_{y_x})$. Note that the $M^{last}_{y_x}$ (the length of the last packet) is removed from the request bound function, since it was already considered in (4).

$$\mathbf{rbf}_{y_x}(t) = IH_{y_x}(t) + BL_{y_x} + C_{y_x} - M^{last}_{y_x} \tag{5}$$

$$IH_{y_x}(t) = \sum_{\Gamma_{y_j} \in hp(\Gamma_{y_x})} \left\lceil \frac{t}{T_{y_j}} \right\rceil \times C_{y_j} \quad (6)$$

$$BL_{y_x} = \max_{\Gamma_{y_j} \in lpe(\Gamma_{y_x})} Mmax_{y_j} \quad (7)$$

The **supply bound function** ($\mathbf{sbf}_{y_x}(t)$) is defined as in (8) using the *Explicit Deadline Periodic* (EDP) resource model [7] that generalizes the periodic resource model for compositional analysis of hierarchical scheduling frameworks. An EDP resource model is given by $\Omega = (\Pi, \Theta, \Delta)$, where Θ is the units of the resource provided within Δ time units (deadline) and with period Π of repetition. This way, mapping to our framework, a server is defined as $\Gamma_{y_x} = (\Pi_{y_x}, \Theta_{y_x}, \Delta_{y_x}) = (T_{y_x}, C_{y_x} - Mmax_{y_x}, RT_{y_x} - Mmax_{y_x})$. Note that we consider the minimum capacity supplied by the server in each instance to be given, in the worst case, by $C_{y_x} - Mmax_{y_x}$, where $Mmax_{y_x}$ is the maximum idle time inserted at the end of each server execution to prevent possible overruns. Also, the response time of the parent server assuming (C_{y_x}) capacity is RT_{y_x}, hence the latest supply of ($C_{y_x} - Mmax_{y_x}$) capacity from the parent server will be $RT_{y_x} - Mmax_{y_x}$. In fact, any packet that starts being transmitted up to $Mmax_{y_x}$ before the end of the server capacity will be able to finish within that same server instance without causing an overrun. However, the analysis will consider the transmission preempted at exactly $C_{y_x} - Mmax_{y_x}$ and continued in the following server instance, thus being safe despite pessimistic. Only the last packet is left out of the higher priority interference computation as shown in (4) and (5). Nevertheless, because of the inserted idle time, this last packet is still guaranteed to fit in the same server instance.

$$sbf_{y_x}(t) = \begin{cases} b\Theta y_x + \max\{0, t - a - b\Pi y_x\}, \\ \qquad t \geq \Delta_{y_x} - \Theta_{y_x} \\ 0, \qquad otherwise \end{cases} \quad (8)$$

where

$$a = (\Pi_{y_x} + \Delta_{y_x} - 2\Theta_{y_x}), \quad b = \left\lfloor \frac{(t - (\Delta_{y_x} - \Theta_{y_x}))}{\Pi_{y_x}} \right\rfloor$$

with

$$\Pi_{y_x} = T_{y_x}, \Theta_{y_x} = C_{y_x} - Mmax_{y_x}$$
$$\text{and } \Delta_{y_x} = RT_{y_x} - Mmax_{y_x}$$

The impact of the inserted idle-time can, in fact, be accounted for in two ways. One is as presented above, where we deduce $Mmax$ from the supply function and keep the real request function. Another alternative was taken in our previous work [12] following the analysis for SIRAP [3] and [4], in which the real supply function is kept and the request function is increased by $Mmax$. Both approaches generate safe worst-case response time estimates. However, the latter approach, because of extending the request function, can potentially include more higher priority interference leading to worst-case response time estimates that are equal or more pessimistic. This is shown further on, in the experiments section.

For some cases when the value of C_{y_x} is close to the value of $Mmax_{y_x}$, then the supply bound function will be very pessimistic since $\Theta_{y_x} = C_{y_x} - Mmax_{y_x}$, while during each server period it is possible to send at least a minimum size packet $Mmin_{y_x}$ in the worst case. We can conclude that $\Theta_{y_x} = \max(C_{y_x} - Mmax_{y_x}, Mmin_{y_x})$. If $\Theta_{y_x} = Mmin_{y_x}$ then Δ_{y_x} should be calculated based on $Mmin_{y_x}$, i.e., $\Delta_{y_x} = RT_{y_x} - Mmin_{y_x}$. In addition and if $\Theta_{y_x} >= Mmin_{y_x}$ and $\Theta_{y_x} < 2Mmin_{y_x}$ then it might happen that the remaining server capacity at time instant t^* when $\mathbf{rbf}_{y_x}(t^*) = \mathbf{sbf}_{P_{y_x}}(t^*)$ in (4), will not be enough to transmit the last packet (less than $Mmin_{y_x}$). One simple solution for this problem is to consider the full preemption analysis for this specific case by setting $M_{y_x}^{last} = 0$ in (4) and (5).

Finally, we do the schedulability assessment. A server hierarchy and associated streams are deemed feasible if the worst-case response time RT_{y_x} of every component Γ_{y_x}, computed as in (4), is at most as long as the component deadline D_{y_x} (9). If this test fails for any component the schedulability analysis fails and the current configuration remains unchanged.

$$\forall \Gamma_{y_x, y=2..NL, x=1..NC_y}, RT_{y_x} \leq D_{y_x} \quad (9)$$

5. ALGORITHM TO DETERMINE THE RESPONSE TIME

The response time of a server Γ_{y_x} (4) is found using the following algorithm, which checks the points of t where $\mathbf{rbf}_{y_x}$ changes [5]. Let $\widehat{CP}_{\mathbf{rbf}_{y_x}}$ be the set of check points that the algorithm uses, being defined as follows:

$$\widehat{CP}_{\mathbf{rbf}_{y_x}} = \bigcup_{\Gamma_{y_j} \in hp(\Gamma_{y_x})} \widehat{cp}_{y_j} \cup T_{y_x}, \quad (10)$$

where $\widehat{cp}_{y_j} = \left\{T_{y_j}, 2T_{y_j}, ..., m_{y_j}T_{y_j}\right\}, m_{y_j} = \lfloor T_{y_x}/T_{y_j} \rfloor$. Therefore, $\widehat{cp}_{y_j}$ contains the set of time instants, associated to the component Γ_{y_j}, during the interval $[0, T_{y_x}]$, where $\mathbf{rbf}_{y_x}(t)$ changes.

The algorithm requires that the set of checking points $\widehat{CP}_{\mathbf{rbf}_{y_x}}$ is sorted in an increasing order. Let $\widehat{CP}^{sorted}_{\mathbf{rbf}_{y_x}}$ be the sorted set, i.e., $\widehat{CP}^{sorted}_{\mathbf{rbf}_{y_x}} = sort(\widehat{CP}_{\mathbf{rbf}_{y_x}})$.

5.1 Algorithm description

Algorithm 5.1 depicts the pseudo-code that allows computing the response times of server components. Its operation is illustrated in Figure 3. The algorithm takes as inputs the component Γ_{y_x} (with $y > 1$), its $\mathbf{rbf}_{y_x}(t)$, the respective set of checkpoints $\widehat{CP}_{\mathbf{rbf}_{y_x}}$ and the $\mathbf{sbf}_{P_{y_x}}(t)$ of its parent component, returning the respective worst case response time RT_{y_x}.

Algorithm 5.1: RESPONSE TIME()

input: $\Gamma_{y_x}, \widehat{CP}_{\mathbf{rbf}_{y_x}(t)}, \mathbf{rbf}_{y_x}(t), \mathbf{sbf}_{P_{y_x}}(t)$
output: RT_{y_x}

1 for each $\left(t_i \in \widehat{CP}_{\mathbf{rbf}_{y_x}}\right)$
2 if $\left(\mathbf{sbf}_{P_{y_x}}(t_i) \geq \mathbf{rbf}_{y_x}(t_i)\right)$
3 $c = \left\lfloor \frac{\mathbf{rbf}_{y_x}(t_i)}{\Theta_{P_{y_x}}} \right\rfloor$
4 if $\left(\mathbf{rbf}_{y_x}(t_i) = c\Theta_{P_{y_x}}\right)$
5 $w_{y_x} = c\Pi_{P_{y_x}} + \Delta_{P_{y_x}} - \Theta_{P_{y_x}}$
6 else
7 $w_{y_x} = \mathbf{rbf}_{y_x}(t_i) + (c+1)\Pi_{P_{y_x}} + \Delta_{P_{y_x}} - (c+2)\Theta_{P_{y_x}}$
8 end if
9 return $(RT_{y_x} = w_{y_x} + M_{y_x}^{last})$
10 end if
11 return -1

Firstly the algorithm finds the earliest instant t_i within the sorted set $\widehat{CP}_{\mathbf{rbf}_{y_x}}^{sorted}$ such that $\mathbf{sbf}_{P_{y_x}}(t_i) \geq \mathbf{rbf}_{y_x}(t_i)$ (lines 1 and 2). When the test of line 2 succeeds, the resulting t_i is an upper bound for the actual response time as computed by Equation 8, thus lines 3-8 compute the correct value. When $\mathbf{rbf}_{y_x}(t_i)$ fully uses the capacity (without inserted idle-time) of the last instance of the parent server, the busy interval of Γ_{y_x} is computed by line 5. Otherwise, it is computed by line 7, which determines the instant corresponding to the fraction of the last parent server instance that is necessary to satisfy the request. c is used as an auxiliary variable in the computation of the correct worst case response time, containing the number of parent server instances fully used by the component under test during its response time interval. If line 2 is satisfied for any checkpoint, the algorithm returns RT_{y_x} as computed in line 9, otherwise it returns a fail code in line 11.

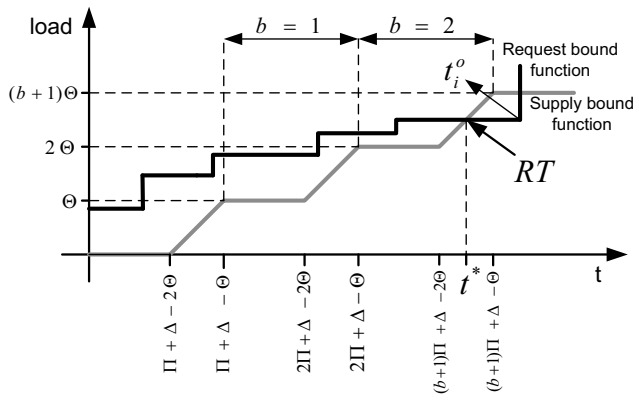

Figure 3: Response time.

5.2 Algorithm complexity

In order to evaluate the time complexity of Algorithm 5.1 we consider the worst case scenario that occurs when the number of checkpoints is maximum, i.e., the periods of higher priority components are relative primes, and all must be computed, i.e., $RT_{y_x} \backsimeq T_{y_x}$. Let nhp_{y_x} be the number of components that belong to $hp(\Gamma_{y_x})$. These components can generate up to $ncp_{y_x} = \sum_{j=1}^{nhp_{y_x}} \left(m_{y_j} + 1\right)$ checkpoints. Therefore, the total computation effort generated during the whole response time interval is, in the worst case, $ncp_{y_x} \times$ ((computational effort of computing $\mathbf{rbf}_{y_x}(t)$ and $\mathbf{sbf}_{P_{y_x}}(t)$) + constant). From Equations 5 and 6, we see that the computational demand for computing **rbf** for a given time instant t is proportional to nhp_{y_x}. Moreover, from Equation 8 we can see that the computational effort for computing **sbf** for a given time t is constant. Therefore, the computational effort of the response time algorithm is $ncp_{y_x} \times ((nhp_{y_x} \times constante) + constant$. Noting that the number of checkpoints is roughly proportional to the number of higher priority components, the computational complexity becomes $O(nhp_{y_x}^2)$, i.e., $O(n^2)$ where n is the number of higher priority components associated with a given server.

6. PROOF-OF-CONCEPT IMPLEMENTATION

As a proof-of-concept implementation, the multi-level hierarchical server-based traffic scheduling framework described above was implemented in the FTT-enabled Ethernet switch [13].

The FTT-enabled Ethernet Switch is a custom FPGA-based Ethernet switch that implements the Flexible Time-Triggered (FTT) paradigm. This paradigm is based on a master/multi-slave architecture. The communication is organized in an infinite succession of Elementary Cycles (ECs). The ECs are divided in two windows – the synchronous window and the asynchronous window. The former is used for synchronous (periodic) communications, triggered by the master via a poll message designated Trigger Message (TM). The asynchronous window is used for asynchronous communications, which are autonomously generated by the slave nodes. In this prototype implementation the multi-level hierarchical server-based traffic scheduling framework is used to manage the asynchronous traffic.

6.1 Implementation overview

The integration of the server framework within the FTT-enabled Ethernet Switch takes advantage of the hardware/software co-design approach used in the switch development. All the low-level server and stream management functions are implemented in hardware (Figure 4) in order to increase the responsiveness of the system. Namely, the scheduling of the asynchronous messages is performed locally, in each output port, using dedicated hardware resources. From the operational and implementation point of view, a stream is a dedicated channel, which can aggregate a configurable number of data flows from different input ports. It comprises exclusive memory resources, reserved in the central memory, and it can be connected to a configurable number of output ports, allowing multicast and broadcast communications, although in the scope of this paper only unicast communications are considered. Each stream can have a configurable set of associated servers, which is independent from port to port. The configuration of the server and stream hierarchy can be freely defined at run-time, without service disruption. The reconfiguration is triggered using dedicated FTT control messages. The amount of resources (e.g. memory, logical blocks) in any FPGA is limited and thus the total

amount of supported servers and streams is limited. No other restrictions to the server topology apply.

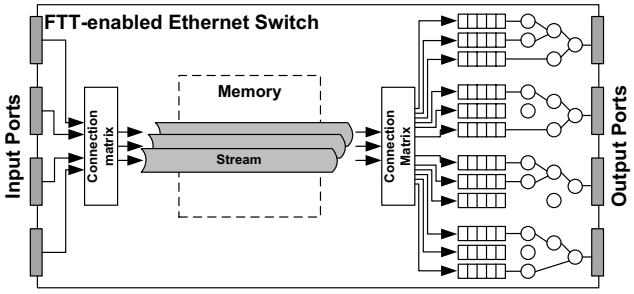

Figure 4: FTT-enabled Ethernet Switch.

The configuration and adaptation of the server hierarchy is managed by a dedicated configuration manager module, which can be executed inside the switch itself or in any node that is part of the network. The configuration manager receives all the server and stream management requests, applies the schedulability analysis algorithm and, if feasible, sends the appropriate commands to reconfigure the switch server scheduling hardware [14].

6.2 Analysis adaptation

As mentioned above, we use the hierarchical server framework to manage the FTT asynchronous window, which is a polling server and it is represented by the component Γ_{1_1} (Figure 1) in the server hierarchy. Figure 5 illustrates the corresponding supply-bound function, as seen by level 2 components. The worst-case scenario happens when a packet transmission request occurs before the beginning of the TM in such a way that it does not fit before the TM and such packet has the maximum allowed size ($Mmax_{1_1}$) thus generating the maximum inserted idle-time (iit). Let LEC be the EC period and LAW the minimum length of the asynchronous window, which are FTT configuration parameters. The corresponding EDP resource model that mimics the asynchronous window supply-bound function is given by $\Omega = (\Pi, \Theta, \Delta)$, with $\Pi_{1_1} = LEC$, $\Theta_{1_1} = LAW - Mmax_{1_1}$ and $\Delta_{1_1} = \Pi_{1_1}$.

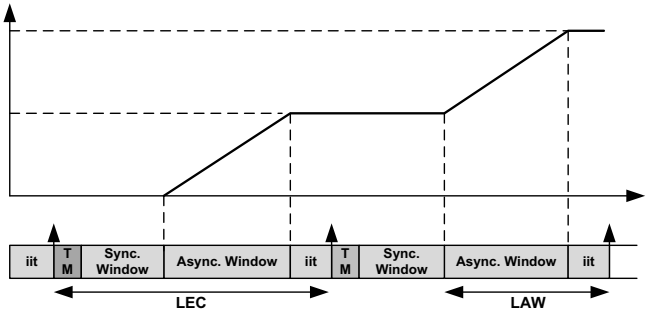

Figure 5: Asynchronous Window.

7. EXPERIMENTAL VALIDATION

This section presents the results obtained from our prototype implementation based on an FTT-enabled Ethernet Switch. Different experiments have been carried out to assess the correctness and degree of pessimism of the analysis, as well as the conformity of the implementation.

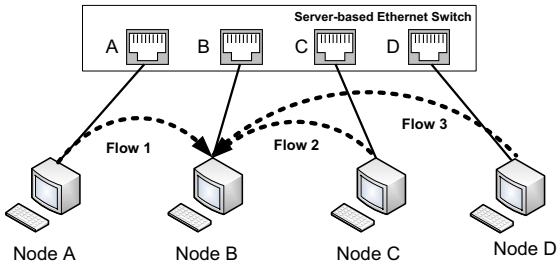

Figure 6: Experimental setup.

Figure 6 shows the experimental setup which represents two distributed control applications whose controllers run on node B. This node also controls the actuators with direct I/O. The applications receive remote sensing via three time-constrained sporadic data flows that are tagged as asynchronous traffic by the FTT switch. These flows are managed by a hierarchy of periodic servers similar to that presented in Figure 1. Each application is handled by one server at the second level of the hierarchy, Γ_{2_1} and Γ_{2_2}, respectively. The latter handles flow 3 (Γ_{3_3}). The former is further divided in two other servers, Γ_{3_1} and Γ_{3_2}, which handle flows 1 (Γ_{4_1}) and 2 (Γ_{4_2}), respectively. The parameters of the servers and flows are shown in Table 1. Flow 1 has two different configurations: a) one single Ethernet packet instances, and b) 3 packets per instance, which are sent in a burst. Finally, the FTT-enabled switch is configured with $LEC = 1000\mu s$ and $LAW = 700\mu s$.

	C	T/Tmit	P	Mmax	Mmin	RT
Γ_{1_1}	700	1000	-	50	25	-
Γ_{2_1}	400	3000	Γ_{1_1}	50	25	1000
Γ_{2_2}	250	2000	Γ_{1_1}	25	25	650
Γ_{3_1}	100	8000	Γ_{2_1}	50	50	3650
Γ_{3_2}	250	4000	Γ_{2_1}	25	25	3550
Γ_{3_3}	25	2200	Γ_{2_2}	25	25	2200
Γ_{4_2}	25	7100	Γ_{3_2}	25	25	7100
$\Gamma_{4_1}a)$	50	25000	Γ_{3_1}	50	50	11550
$\Gamma_{4_1}b)$	150	25000	Γ_{3_1}	50	50	19600

Table 1: Experiment 1 – hierarchy parameters in μs

7.1 Experiment 1

In this experiment we used configuration a) of flow 1 (Γ_{4_1}) and we logged the response times of 37000 single packet instances corresponding to approximately 15min of consecutive operation. We used the parameters for servers and streams shown in Table 1. We started by computing $Mmax$ and $Mmin$ along the hierarchy, from bottom to top, thus starting from the streams Γ_{4_1}, Γ_{4_2} and Γ_{3_3} and up to the top server Γ_{1_1}. The resulting values are also shown in the table.

Then we carried out the schedulability analysis from top to bottom, computing the worst-case response times RT for each server and stream, also shown in Table 1, and comparing with the respective deadlines. The analysis deemed this

hierarchy schedulable with the worst-case response time for flow 1 being upper bounded to 11550μs.

The actual histogram of the logged response-times for all instances is shown in Figure 7. The maximum observed response time is 8307μs, which is below the upper bound provided by the analysis. The difference between both values can be explained by two main reasons. On one hand, the experiment might have not generated the real worst-case situation and on the other hand, the analysis still includes a few pessimistic assumptions with respect to the worst-case scenario such as the actual execution pattern of the servers and the inserted idle-time.

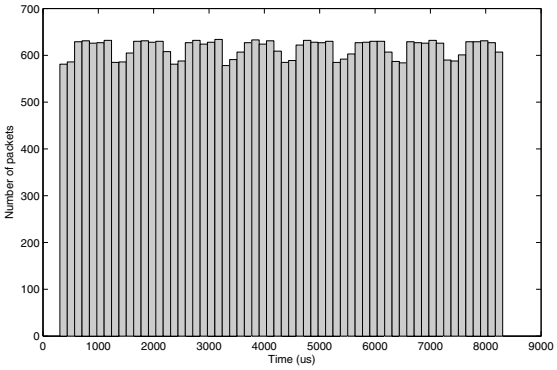

Figure 7: Response time histogram of $\Gamma_{4_1} a$).

7.2 Experiment 2

This experiment used configuration b) of flow 1 (Γ_{4_1}) and we logged the response times of 130000 multipacket instances corresponding to approximately 54min of consecutive operation. Again, we used the parameters for servers and streams shown in Table 1. With respect to the previous experiment there was no change in $Mmax$ and $Mmin$. However, each instance of flow 1 in this configuration required two instances of the parent server Γ_{3_1} to be transmitted.

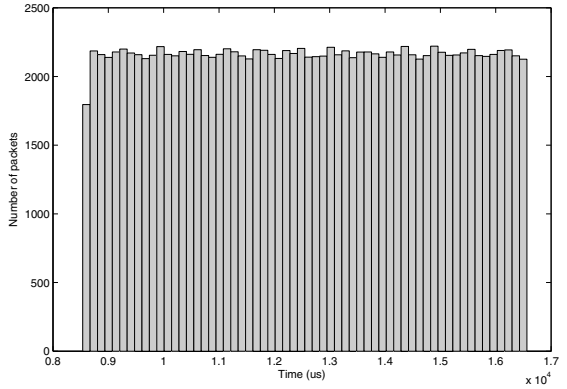

Figure 8: Response time histogram of $\Gamma_{4_1} b$).

Again, the schedulability analysis deemed this hierarchy schedulable. The only change with respect to the previous case was the response time for flow 1 being now upper bounded to 19600μs. Figure 8 shows the histogram of the observed response times of all instances. The maximum observed response time was 16564μs, which is still below the upper bound provided by the analysis.

7.3 Experiment 3

Finally, we carried out an experiment to compare the level of pessimism embodied in our analysis and in the analysis in [12]. For this purpose we consider the set of components in Table 2 and we analyze such hierarchy with both analysis. The results are also listed in the same table and we can see that both analysis guarantee the schedulability of the set. For this example, the response times obtained by our analysis ($RT1$) are substantially shorter than those obtained with the analysis in [12] ($RT2$). However, we cannot guarantee a similar level of improvement in the general case, due to dependency on the actual server hierarchy and streams, but it is guaranteed that our analysis will always provide equal or better results than the previous one in [12].

	C	T/Tmit	P	Mmax	Mmin	RT1	RT2
Γ_{1_1}	600	1000	-	150	50	-	-
Γ_{2_1}	350	3000	Γ_{1_1}	150	80	1025	1725
Γ_{2_2}	125	2000	Γ_{1_1}	100	50	825	975
Γ_{3_1}	200	8000	Γ_{2_1}	100	80	6675	7475
Γ_{3_2}	200	7500	Γ_{2_1}	150	100	3775	7425
Γ_{3_3}	150	15000	Γ_{2_2}	100	50	4750	12950
Γ_{4_2}	250	50000	Γ_{3_2}	150	100	18625	44875
Γ_{4_1}	280	35000	Γ_{3_1}	100	80	22555	31355

Table 2: Experiment 3 – hierarchy parameters in μs

8. CONCLUSIONS

Current Real-Time Ethernet protocols do not allow efficient server-based scheduling policies as those developed for CPU scheduling, impairing the use of component-oriented design technologies in networked embedded systems. Such limitation fostered the recent proposal of a multi-level hierarchical server architecture for Ethernet switches. This architecture provides a dynamically reconfigurable hierarchical composition of servers that allows to divide the network bandwidth in a hierarchical way, creating virtual channels with temporal isolation among them, thus supporting composability in the time domain. This paper presents a novel response-time based schedulability analysis framework that takes into consideration the impact of the hierarchy and of the server operational mechanisms into the schedulability. This test, which complexity is shown to be approximately $O(n^2)$ for each task, with n being the number of higher priority components, is part of an admission control for dynamic resource reservation. In addition to the analytical framework, the paper also presents a prototype implementation based on an FTT-enabled Ethernet Switch that validates the analysis and the multi-level hierarchical server architecture and shows that the our analysis reduces the pessimism of previous analysis developed for similar frameworks.

9. ACKNOWLEDGMENTS

This work was partially supported by the iLAND project, call 2008-1 of the EU ARTEMIS JU Programme, by the Eu-

ropean Community through the ICT NoE 214373 ArtistDesign and program FEDER through "Programa Operacional Factores de Competitividade – COMPETE", by the Portuguese Government through "FCT – Fundação para a Ciência e a Tecnologia" in the scope of project FCOMP-01-0124-FEDER-007220 and Ph.D. grant - SFRH/BD/32814/2006 and by the Swedish Foundation for Strategic Research (SSF), the Swedish Research Council.

10. REFERENCES

[1] Ethernet Powerlink - online information. http://www.ethernet-powerlink.org/.

[2] L. Almeida and p. Pedreiras. Scheduling within temporal partitions: response-time analyisis and server design. In *Proceedings of the Fourth ACM International Conference on Embedded Software*, September 2004.

[3] M. Behnam, T. Nolte, and R. Bril. SIRAP: A synchronization Protocol for Hierarchical Resource Sharing in Real-Time Open Systems. In *Proceedings of the Seventh ACM International Conference on Embedded Software*, October 2007.

[4] M. Behnam, T. Nolte, and R. Bril. Refining SIRAP with a dedicated resource ceiling for self-blocking. In *Proceedings of the Nineth ACM International Conference on Embedded Software*, October 2009.

[5] E. Bini and G. Buttazzo. The space of rate monotonic schedulability. In *Proceedings of IEEE International Real-Time Systems Symposium*, December 2002.

[6] L. Deng and J. Liu. Scheduling real-time applications in an open enviroment. In *Proceedings of IEEE International Real-Time Systems Symposium*, December 1997.

[7] A. Easwaran, M. Anand, and I. Lee. Compositional Analysis Framework using EDP Resource Models. In *Proceedings of IEEE International Real-Time Systems Symposium*, December 2007.

[8] T. Kuo and C. Li. A fixed-priority-driven open enviroment for real-time applications. In *Proceedings of IEEE International Real-Time Systems Symposium*, December 1999.

[9] J. Loeser and H. Haertig. Low-Latency Hard Real-Time Communication over Switched Ethernet. In *Proceedings of IEEE Euromicro Conference on Real-Time Systems*, 2004.

[10] P. Pedreiras and L. Almeida. Combining Event-triggered and Time-triggered Traffic in FTT-CAN: Analysis of the Asynchronous Messaging System. In *Proceedings of IEEE Workshop on Factory Communication Systems*, September 2000.

[11] PROFInet. Real-Time PROFInet IRT. http://www.profibus.com/pn.

[12] R. Santos, M. Behnam, T. Nolte, P. Pedreiras, and L. Almeida. Schedulability Analysis for Multi-level Hierarchical Server Composition in Ethernet Networks. In *Presentation at the Workshop on Real-Time Networks*, July 2010.

[13] R. Santos, P. Pedreiras, L. Almeida, A. Vieira, T. Nolte, R. Marau, and A. Oliveira. Flexible, Efficient and Robust Real-Time Communication with Server-based Ethernet Switching. In *Proceedings of IEEE Workshop on Factory Communication Systems*, May 2010.

[14] R. Santos, A. Vieira, R. Marau, P. Pedreiras, A. Oliveira, L. Almeida, and T. Nolte. Improving the efficiency of Ethernet switchess for real-time communication. In *Proceedings of Frist International Workshop on Adaptive Resource Management*, April 2010.

[15] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of IEEE International Real-Time Systems Symposium*, December 2003.

[16] Q. Wang, S. Gopalakrishnan, X. Liu, and L. Sha. A Switch Design for Real-Time Industrial Networks. In *Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium*, 2008.