Schedulability Analysis for Multi-level Hierarchical Server Composition in Ethernet Switches

Rui Santos, Paulo Pedreiras,
IEETA / University of Aveiro
Aveiro, Portugal
{rsantos,pbrp}@ua.pt

Moris Behnam, Thomas Nolte
MRTC / Mälardalen University
Västerås, Sweden
{moris.behnam,thomas.nolte}@mdh.se

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

Abstract

The FTT-enabled (Flexible Time-Triggered) Ethernet Switch provides flow-based dynamic scheduling that allows to handle bursty traffic in a bandwidth efficient way. For that, this switch uses adaptive resource-reservation, associating servers to flows or groups of flows. This way, flows have a guaranteed, but bounded, access to the communication resources. These servers can take up a compositional multi-level hierarchy and they can be adapted on-line to make a better use of the available bandwidth. To assure a continued real-time behavior, the FTT-enabled Ethernet Switch integrates an admission control mechanism, which screens all adaptation and/or reconfiguration requests. Whenever such requests may compromise the flow timeliness or exceed the available memory, they are rejected. This paper focuses on the flow timeliness verification, only, providing a response-time based schedulability analysis that permits assessing the schedulability of a hierarchical composition of servers and flows. [1]

1 Introduction

Many current embedded applications are complex entities structured in components and usually assuming a hierarchical composition. However, these applications can suffer limitations in accessing resources due to lack of frameworks that provide adequate resource access control in such complex systems. For example, in the network domain, integrating different applications with different communication requirements under real-time constraints can generate problems of resource allocation (bandwidth) and temporal isolation between streams or across applications. One recent paradigm that favors the development of frameworks to support hierarchical structures is component-based design in which applications are built by composing diverse components developed separately. The benefits range from reduction of design complexity to more efficient resource sharing, satisfying individual service requirements of each component and enforcing mutual temporal isolation.

Architectures based on servers that act as containers in the temporal domain have been recognized as an effective means to enable such kind of resource sharing [1] and they can be the basis for resource partioning and virtualization, supporting the separation between the applications and the hardware platform on which they will execute. Following this trend, the FTT-enabled (Flexible Time-Triggered) Ethernet Switch [2] provides a framework to carry out hierarchical composition of servers that divides the network resource in an efficient way and allows an easy and natural mapping of the applications onto the network. Moreover, the use of servers for flow management allows handling heterogeneous kinds of traffic with arbitrary arrival patterns and with temporal isolation.

This paper presents an extension of previous work in this framework, particularly that reported in [3], which defined an adaptation and reconfiguration protocol that allowed adding, removing and modifying servers and the associated asynchronous flows. An on-line admission control using a light utilization-based schedulability analysis enforced continued timeliness even during changes. The traffic scheduling followed the *blocking-free non-preemptive model*, which applies when the traffic is scheduled in cycles, within partitions, and before the start of the respective partition (or window), adhering strictly to the partition duration. This implies one aspect, there is an extra delay since an arriving packet might have to wait up to one cycle to be considered by the scheduler for possible transmission. An alternative that improves the latency of the switch is to enable the scheduling during the partition, executing it whenever a packet arrives. In this case, the previous analysis does not

[1]This work was partially supported by the iLAND project, call 2008-1 of the EU ARTEMIS JU Programme, by the European Community through the ICT NoE 214373 ArtistDesign and by the Portuguese Government through the FCT project HaRTES - PTDC/EEA-ACR/73307/2006 and Ph.D. grant - SFRH/BD/32814/2006.

hold and the blocking caused by the transmission of non-preemptive packets of lower priority servers must be taken into account. Therefore, this paper presents a schedulability analysis based on response time for a multi-level hierarchical server composition that handles the asynchronous flows within the FTT framework and considers the blocking referred above.

The remainder of the paper is organized as follows. Section 2 presents an overview of schedulability analysis for hierarchical scheduling frameworks. Section 3 introduces the basics of the FTT-enabled Ethernet Switch and describes its integration with the server-based traffic scheduling. Section 4 presents the schedulability analysis and an algorithm for determining the response time. Finally, Section 5 concludes the paper and addresses future work.

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 servers category called *traffic shapers* [4], which have the purpose of limiting 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.

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 [5] and Ethernet Powerlink [6]). This is a trivial composition of several PS that results in an inefficient use of the network bandwidth. Other protocols, such as [4], implement traffic shapers in the end nodes that behave similarly to a DS. However, due to infrastructural limitations, none of these protocols supports arbitrary server policies nor their hierarchical composition and dynamic adaptation or creation/removal.

Another related area, despite typically considering preemptive task scheduling, is that of general hierarchical scheduling frameworks (HFS). Deng and Liu [7] began proposing two levels HFS for open systems, where subsystems may be developed and validated independently. Kuo and Li [8] introduce for such two levels a schedulability analysis based on Fixed Priority Scheduling (FPS) with a global scheduler. Shin and Lee [9] present a generic scheduling interface model in order to construct hierarchical scheduling frameworks. Almeida and Pedreiras [10] 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.* [11] generalize the periodic resource model for compositional analysis of hierarchical scheduling frameworks.

Finally, another related area is that of synchronization protocols in HFS. For example, SIRAP [12] addresses CPU resource sharing among several subsystems that execute within servers and it proposes inserting idle-time (iit) whenever the remaining capacity is not enough to execute an access to a shared resource. However, only two level HFS are addressed, while in this paper we seek explicitly the support to multi-level HFS.

3 FTT-enabled Ethernet Switch

The FTT-enabled Ethernet Switch was created in the scope of FTT paradigm [13]. The FTT paradigm is a master multi-slave communication protocol, where a Master node coordinates the transmissions of other nodes (Slaves) by means of the periodic transmission of a Trigger Message (TM) that contains the schedule for a fixed-duration time slot, designated Elementary Cycle (EC) (Figure 1). The communication is organized in an infinite succession of such Elementary Cycles (ECs).

The FTT framework defines three traffic classes: 1) periodic real-time messages triggered by the master (referred to as *synchronous* since their transmission is synchronized with the master traffic scheduler); 2) aperiodic or sporadic real-time traffic, autonomously triggered by the nodes; and 3) non real-time traffic (classes 2 and 3 are referred to as *asynchronous*). The EC is organized in two windows, synchronous and asynchronous, which convey the corresponding traffic classes.

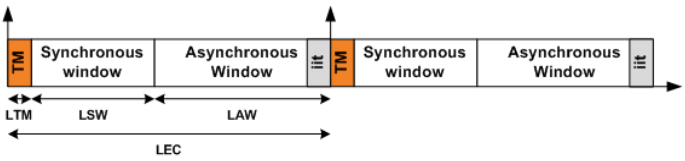

Figure 1: Elementary Cycles

At the begining of each EC the switch broadcasts the TM to all slave nodes, identifying which messages should be transmitted. Synchronous messages are always polled by the TM. In the particular case of the FTT-Enabled Ethernet Switch, no polling for the asynchronous traffic is necessary since the switch is aware of the EC structure (Figure 2) and has a complete control of the message forwarding procedure. Therefore, the asynchronous messages may be sent by the respective sources at arbitrary instants since the switch is able to queue them in dedicated memory pools and transmit them to the respective destinations only during the asynchronous windows. Appropriate scheduling mechanisms, e.g. servers, may be used to schedule the queued asynchronous messages.

The autonomous confinement of messages by the FTT-enabled Ethernet Switch is one of the distinctive features

of this protocol with respect to its predecessors, namely the FTT-SE protocol. This latter protocol relies on Commercial Of-The-Shelf (COTS) Ethernet switches and thus all nodes have to comply with the protocol, i.e., have to integrate a specific device driver, to ensure that the message transmissions occur only at adequate time instants. This is an important limitation since legacy nodes cannot be part of the network. Additionally, the management of the asynchronous traffic is less efficient since this type of traffic has also to be scheduled by the master node and an explicit signaling mechanism by nodes informing the master about the ready asynchronous traffic is required. Finally, the tight control of the message forwarding combined with the awareness of the message requirements also allows the switch to detect failures in the time domain, such as nodes that transmit asynchronous messages at higher rates than the ones declares or that send synchronous messages that where not scheduled by the Master node, preventing its transmission.

Summing up, the FTT-enabled Ethernet Switch provides the following features:

1. online admission control, dynamic QoS management and arbitrary traffic scheduling policies;
2. high system integrity with unauthorized real-time messages being eliminated at the switch input ports;
3. asynchronous traffic autonomously triggered by the nodes, with arbitrary arrival patterns;
4. high configurability: fully synchronous mode, adjustable mixed synchronous/asynchronous mode and fully asynchronous mode;
5. a standard node can take advantage of the real-time services simply negotiating with the switch the creation of a server, i.e., a virtual channel (the negotiation can even be done by a third party node);
6. a standard node can readily transmit non-real-time traffic using a background server thus not interfering with the real-time traffic.

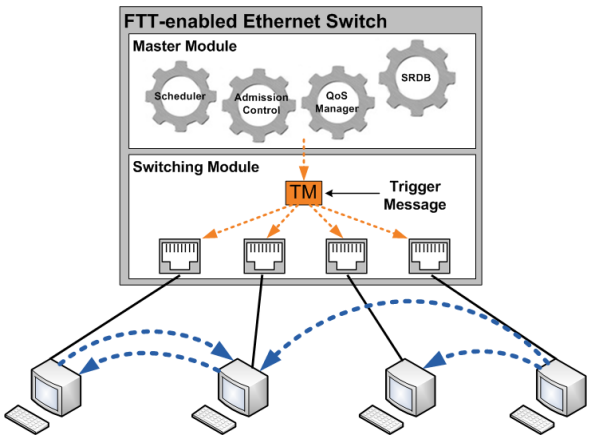

Figure 2: FTT-enabled Ethernet Switch.

3.1 FTT EC structure as composition of servers

As mentioned above, in the FTT-enabled Ethernet Switch the traffic is divided in synchronous and asynchronous classes, associated with disjoint windows that fill in the usable part of the EC. These windows appear once in each EC (Figure 3) and have a bounded size (LSW and LAW, respectively). Note that LSW correspond to an upper bound (the synchronous traffic may use up to LSW in each EC) while LAW refers to a lower bound (asynchronous traffic can use at least LAW in each EC), since the asynchronous window reclaims the bandwidth not used by the synchronous one. Using a server terminology, the synchronous window is associated with a server characterized by a period $T_{SW} = T_{EC}$ and a (maximum) capacity $C_{SW} = LSW$, while the asynchronous window is associated with a server with a (minimum guaranteed) capacity of $C_{AW} = LAW$ and a period $T_{AW} = T_{EC}$. Note that LEC, LSW and LAW are FTT configuration parameters that can be tuned to suit the global application needs.

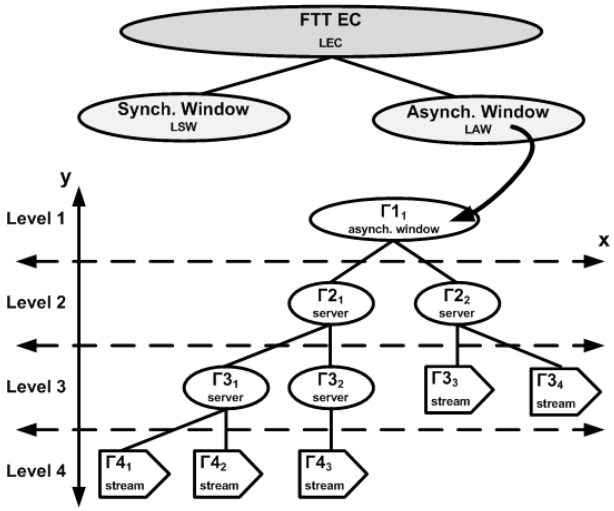

Figure 3: Server Hierarchy.

3.2 Hierarchical Server Composition in the scope of the FTT-enabled Ethernet Switch

In recent work [14] [13], it was proposed to integrate hierarchical server composition on the FTT framework to manage the asynchronous traffic. As illustrated in Figure 3, asynchronous message streams (or streams, for short) are handled by servers. On its hand servers may also depend on other servers. Each server should have enough resources to handle its childs, should they be streams or other servers. Servers and streams are abstracted by components. At each level a component Γ_{y_x} is identified by both index y and x. The index y identifies the level in the hierarchy and the index x identifies the component inside that level. 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 underlying FTT framework puts some important constraints on the server operation that affect the system schedulability, namely: 1) Ethernet does not permit packet preemption thus preemption is not allowed. Consequently, packets of high priority components may be blocked by ongoing transmissions of lower priority ones. 2) Overruns are not allowed by design, since the capacity is strictly enforced (the switch does not initiate a message transmission that does not fit in the remaining capacity). The combination of 1) and 2) results in a potential appearance of idle time at the end of each server instance, whenever the remaining capacity is not enough to transmit the following queued packet. Despite negative from the schedulability point of view, this *modus operandi* enforces a strict temporal isolation between all components all the way through the top of the hierarchy. Thus, ECs are completely regular and the TM does not suffer any interference from packets managed by server components inside the asynchronous window.

4 Schedulability analysis

Assuring a continued real-time behavior requires the execution of an admission control procedure every time the message set is changed. In the basis of this admission control procedure there is a schedulabilty test. In [3] it is presented an on-line admission control using a light utilization-based schedulability analysis. However, that analysis is based on the *blocking-free non-preemptive model*, which requires that the traffic has to be scheduled in cyclic fashion, within partitions, and before the start of the respective partition (or window). This paper removes such dependency, supporting an unrestricted activation model, i.e. streams enter the scheduling process immediately after being received by the switch. Although more complex, this operation mode reduces the stream forwarding latency, which is an important merit factor in many application scenarios.

4.1 Traffic and resource model

The asynchronous streams are at the end of the hierarchy (Figure 3) and they are characterized in (1) through the sporadic real-time model, where C_{y_x} is the maximum transmission time and $Tmit_{y_x}$ represents the respective minimum interarrival time. $Mmax_{y_x}$ and $Mmin_{y_x}$ is the transmission time of the largest and smallest packet, respectively, transmitted by this stream. P_{y_x} identifies the parent server, i.e, the server to which the stream is connected to and RT_{y_x} its computed response time.

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

On the other hand, inside the asynchronous window the servers (components) assume a hierarchical composition with multi-level, forming several branches. The individual server Srv_{y_x} (2) is characterized by its capacity C_{y_x}, its replenishment period T_{y_x}, and a few figures 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 Srv_{y_x} is characterized by a parent server P_{y_x} and its computed response time RT_{y_x}. Note that despite the similarity between the characterization of servers and streams, there is a fundamental difference since the stream implies actual transmission time that uses the capacity of the respective server.

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

4.2 Schedulability algorithm

As referred before, the servers 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 by inserting idle-time (iit), called *self-blocking* in the scope of SIRAP [12], 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 4). Therefore, the maximum inserted idle-time (iit) that a server component Γ_{y_x} can suffer is equal to the maximum packet transmission time managed by this server ($Mmax_{y_x}$). On the other hand, $Mmax_{y_x}$ 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. Moreover, the $Mmin_{y_x}$ is used to know the maximum memory required in each branch, but this subject is out of the scope in this paper. This way, before performing any change to the message set it is necessary to assess its impact in the parameters $Mmax$ and $Mmin$ along the hierarchy. Therefore, the schedulability algorithm presented in this section is executed in two phases.

4.2.1 Schedulability algorithm - first phase

The first phase of the algorithm simulates the requested change in the hierarchy and by going from the bottom to the top aims to find the $Mmax$ and $Mmin$ packet transmission time in each branch. This means, for instance, at the end of this phase, the component that manages the asynchronous window Γ_{1_1} will have the maximum ($Mmax$) and

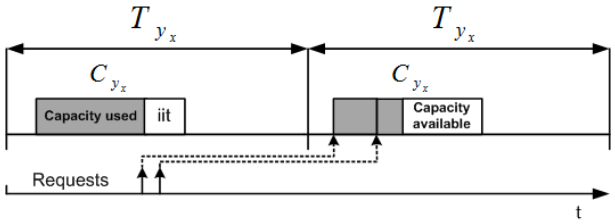

Figure 4: Inserting idle-time (iit) to enforce servers capacities

the minimum ($Mmin$) packet transmission time among all the asynchronous streams transmitted in that window.

Example. As an example, consider the components shown in Figure 3. Assume that the streams have the following $Mmax$ and $Mmin$, respectively: $\Gamma_{3_3}(120000, 8000)$, $\Gamma_{3_4}(121000, 8000)$, $\Gamma_{4_1}(117000, 8000)$, $\Gamma_{4_2}(118000, 8000)$, $\Gamma_{4_3}(119000, 8000)$. Given such scenario, after the first phase of the schedulability algorithm, the servers (components) in the hierarchy inherit the maximum $Mmax$ and the minimum $Mmin$ of their children thus resulting in $\Gamma_{3_1}(118000, 8000)$, $\Gamma_{3_2}(119000, 8000)$, $\Gamma_{2_1}(119000, 8000)$, $\Gamma_{2_2}(121000, 8000)$, and finally $\Gamma_{1_1}(121000, 8000)$.

4.2.2 Schedulability algorithm - second phase

The second phase consists in verifying, from the top to the bottom of the hierarchy, the schedulability of each component and consequently the schedulability of the whole system. For this purpose, a local schedulability analysis under FPS, presented in [9], is used:

$$\forall \Gamma_{y_x} \exists t : 0 < t \leq T_{y_x}, \mathbf{rbf}_{y_x}(t) \leq \mathbf{sbf}_{P_{y_x}}(t), \quad y = 2..NL \text{ and } x = 1..NC_y, \tag{3}$$

where, $\mathbf{rbf}_{y_x}(t)$ denotes the request bound function of the component Γ_{y_x} that, for each instant t, quantifies the maximum load submitted to the parent component P_{y_x} by the component itself together with interference of its high priority components and also together with blocking of low priority components. On the other hand, $\mathbf{sbf}_{P_{y_x}}(t)$ is the supply bound function associated to the parent component of Γ_{y_x} that computes the minimum bandwidth supply provided to its children, in each instant t. Consequently, the worst case response time of a component Γ_{y_x} is given by the first intersection between the $\mathbf{rbf}_{y_x}$ and $\mathbf{sbf}_{y_x}$, and it is described as follows:

$$RT_{y_x} = \text{ shortest } t^* : \mathbf{rbf}_{y_x}(t^*) = \mathbf{sbf}_{P_{y_x}}(t^*) \tag{4}$$

After a suitable schedulability analysis, the requested change is introduced in the hierarchical structure and the simulated configuration performed in the first phase takes effect. On the other hand, if the schedulability analysis fails the old configuration remains unchanged.

Supply bound function. In order to define the $\mathbf{sbf}_{y_x}(t)$, we use the *Explicit Deadline Periodic* (EDP) resource model [11] 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 within Δ time units (deadline) and with period Π of repetition. This way, mapping to our framework, a component is defined as $\Gamma_{y_x} = (\Pi_{y_x}, \Theta_{y_x}, \Delta_{y_x}) = (T_{y_x}, C_{y_x}, RT_{P_{y_x}})$, where the $RT_{P_{y_x}}$ is the response time of the parent component. However, for the first component in the hierarchy (Γ_{1_1}), the asynchronous window, we consider the Δ equal to the capacity of the window (C_{1_1}), resulting that $\Gamma_{1_1} = (\Pi_{1_1}, \Theta_{1_1}, \Delta_{1_1}) = (T_{1_1}, C_{1_1}, C_{1_1})$. Therefore, according to the EDP model, and assuming the same notation, the $\text{sbf}_{y_x}(t)$ is defined as follows:

$$sbf_{\Gamma_{y_x}}(t) = \begin{cases} b\Theta y_x + \max\{0, t - a - b\Pi y_x\}, \\ \qquad t \geq \Delta_{y_x} - \Theta_{y_x} \\ 0, \quad otherwise \end{cases} \tag{5}$$

where $a = (\Pi_{y_x} + \Delta_{y_x} - 2\Theta_{y_x})$ and $b = \lfloor (t - (\Delta_{y_x} - \Theta_{y_x}))/\Pi_{y_x} \rfloor$. Moreover, $\Pi_{y_x} = T_{y_x}$, $\Theta_{y_x} = C_{y_x}$, $\Delta_{y_x} = C_{y_x}$ when $y = 1$ or $\Delta_{y_x} = RT_{P_{y_x}}$ when $y > 1$.

Request bound function. The $\mathbf{rbf}_{y_x}(t)$ of a component Γ_{y_x} is given by the following equation, similarly to the analysis of SIRAP [12]:

$$\mathbf{rbf}_{y_x}(t) = C_{y_x} + IS_{P_{y_x}}(t) + IH_{y_x}(t) + IL_{y_x}, \tag{6}$$

$$IS_{P_{y_x}}(t) = \left\lceil \frac{t + Mmax_{P_{y_x}}}{T_{P_{y_x}}} \right\rceil \times Mmax_{P_{y_x}}, \tag{7}$$

$$IH_{y_x}(t) = \sum_{\Gamma_{y_j} \in hp(y_x)} \left\lceil \frac{t}{T_{y_j}} \right\rceil \times C_{y_j} \tag{8}$$

$$IL_{y_x} = \max_{\Gamma_{y_j} \in lp(y_x)} Mmax_{y_j}, \tag{9}$$

where $IS_{P_{y_x}}(t)$ is the maximum inserted idle-time (self-blocking) from the parent component and it is modeled by a virtual component of high priority with a period $T_{P_{y_x}}$, a capacity equal to $Xmax_{P_{y_x}}$ and a phase equal to $Xmax_{P_{y_x}}$. $IH_{y_x}(t)$ is the interference from the components with higher priority in the same level and in the same branch, IL_{y_x} is the blocking from components with lower priority.

Despite the similarities with the analysis in [12], this new approach introduces the impact of our multi-level hierarchi-

cal framework in which the impact of the inserted idle-time in the child components is accounted for in $IS(t)$.

4.3 Computing the response time

The response time of each component Γ_{y_x} (with $y > 1$) can be obtained solving iteratively equation (4), and making use of the inverse of the supply bound function as follows:

$$RT_{y_x} = \text{ earliest } t^* : t^* = \mathbf{sbf}^{inv}_{P_{y_x}}(\mathbf{rbf}_{y_x}(t^*)) \qquad (10)$$

A simpler but less tight upper bound for the response time of each component can be obtained considering a linear lower bound to the supply bound function, also proposed in [9]. This linear lower bound is depicted in Figure 5 and results in $\mathbf{sbf\_lb}_{y_x} = (t-(\Pi_{y_x}-\Delta_{y_x}-2\Theta_{y_x}))\alpha$. Therefore the response time upper bound ($RT\_up_{y_x}$) can be obtained replacing $\mathbf{sbf}_{y_x}$ in (10) by $\mathbf{sbf\_lb}_{y_x}$.

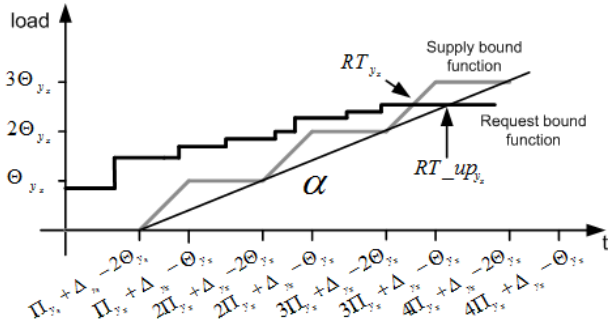

Figure 5: Response time

5 Conclusions

Component-based design is a powerful design paradigm to address the growing complexity of embedded applications. Moreover, server-based scheduling is an effective means to deploy component-based applications, particularly when organized in a hierarchical framework. In this paper, we addressed the case of multi-level hierarchical server-based scheduling within Ethernet switches using a specific switch, namely the FTT-enabled Ethernet Switch. We have developed a schedulability analysis that allows verifying whether a given composition of servers is schedulable. This analysis applies to cases in which servers might experience blocking caused by on-going packet transmissions associated to lower priority servers, and it complements the work in [3] that applies when such blocking is eliminated using the *blocking-free non-preemptive model*. This latter approach, however, presents a longer switching latency, which might not be desirable. A full comparisons and analysis of these two approaches will be carried out in future work.

References

[1] I. Shin and I. Lee, "Compositional real-time scheduling framework with periodic model," *ACM Trans. Embed. Comput. Syst.*, 2008.

[2] R. Santos, R. Marau, A. Oliveira, P. Pedreiras, and L. Almeida, "Designing a Costumized Ethernet Switch for Safe Hard Real-Time Communication," in *Proceedings of the 7th IEEE Workshop on Factory Communication Systems*, May 2008.

[3] R. Santos, A. Vieira, R. Marau, P. Pedreiras, A. Oliveira, L. Almeida, and T. Nolte, "Improving the efficiency of Ethernet switches for real-time communication," in *Proceedings of the 1st International Workshop on Adaptive Resource Management*, April 2010.

[4] J. Loeser and H. Haertig, "Low-Latency Hard Real-Time Communication over Switched Ethernet," in *Proc. of the 16th IEEE Euromicro Conf. on Real-Time Systems*, 2004.

[5] PROFInet, "Real-Time PROFInet IRT," http://www.profibus.com/pn.

[6] "Ethernet Powerlink - online information," http://www.ethernet-powerlink.org/.

[7] L. Deng and J. Liu, "Scheduling real-time applications in an open enviroment," in *Proceedings of the 18th IEEE Inter. Real-Time Systems Symposium*, December 1997.

[8] T. Kuo and C. Li, "A fixed-priority-driven open enviroment for real-time applications," in *Proc. of the 20th IEEE International Real-Time Systems Symposium*, December 1999.

[9] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *Proceedings of the 24th IEEE International Real-Time Systems Symposium*, December 2003.

[10] L. Almeida and P. Pedreiras, "Scheduling within temporal partitions: response-time analyisis and server design," in *Proceedings of the 4th ACM International Conference on Embedded Software*, September 2004.

[11] A. Easwaran, M. Anand, and I. Lee, "Compositional Analysis Framework using EDP Resource Models," in *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, December 2007.

[12] M. Behnam, I. Shin, T. Nolte, and M. Nolin, "SIRAP: A synchronization Protocol for Hierarchical Resource Sharing in Real-Time Open Systems," in *Proceedings of the 7th ACM International Conference on Embedded Software*, October 2007.

[13] R. Santos, R. Marau, A. Vieira, P. Pedreiras, A. Oliveira, and L. Almeida, "A Synthesizable Ethernet Switch with Enhanced Real-Time Features," in *Proceedings of the 35th IEEE Industrial Electronics Society*, November 2009.

[14] R. Santos, A. Vieira, R. Marau, P. Pedreiras, A. Oliveira, L. Almeida, and T. Nolte, "Implementing Server-based Communications within Ethernet Switches," in *Proceedings of 2nd International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, December 2009.