Improving the efficiency of Ethernet switches for real-time communication

R. Santos, A. Vieira, R. Marau, P. Pedreiras, A. Oliveira
DETI / IEETA
Universidade de Aveiro, Portugal
{rsantos,alexandrevieira,marau,pbrp,arnaldo}@ua.pt

Luis Almeida
IEETA - DEEC / University of Porto
4200-465 Porto, Portugal
lda@fe.up.pt

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

Abstract

The growth of connectivity in general, including in networked embedded systems, is continuously increasing the amount and diversity of information that needs being exchanged among nodes. The arrival pattern of such traffic is also frequently bursty and jittered, negatively impacting the system performance. In this paper we focus on switched Ethernet and we aim at providing flow-based dynamic scheduling that allows handling bursty and jittered traffic in a bandwidth efficient way. For that we use adaptive resource-reservation, associating servers to flows or groups of flows that contain the impact that a misbehaving flow can have. These servers can be adapted on-line to make a better use of the available bandwidth. [1]

1 Introduction

The trend towards integration of all kinds of gadgets, devices, machines, systems, etc, is continuously generating more traffic and with predominance of arbitrary arrival patterns with occasional peaks. This trend is clear in the Internet [1] where traffic peaks are being handled by resource over-provisioning even if the average load remains way below the network full capacity. This situation is mainly caused by inefficient traffic management in conventional routers, based on handling packets individually. For example, when momentarily overloaded, a router discards packets somewhat randomly, potentially causing strong throughput penalties, e.g., when several TCP/IP transmissions are affected.

To improve efficiency for the same throughput and handle peaks with less resources, Internet specialists are proposing *flow management*. The first packet in a flow is handled as normal but it is identified, characterized and its forwarding parameters are stored in a hash table. The following packets of each flow are then quickly dispatched. When a router is overloaded, it adjusts each flow rate at its input according to the total available bandwidth. This can be done selectively discarding packets of some flows while protecting the throughput of other ones that have higher priority. In summary, this approach decreases and stabilizes the packet forwarding time, reduces packet losses and overall improves the network bandwidth utilization.

Shifting the focus to real-time networks we often find the same situation in which bandwidth is over-provisioned to guarantee upper limits on worst-case network latencies leading to an overall poor efficiency in resource usage. This has been particularly true in real-time switched Ethernet networks with aperiodic/sporadic traffic where the aggregated throughput is frequently strangled in order to enforce determinism (e.g., Ethernet Power Link and Ethernet/IP).

The time-triggered paradigm is a possible solution (e.g. TTEthernet and PROFINET-IRT), which is particularly efficient with globally synchronous periodic communication that can be efficiently packed in tight schedules that maximize bandwidth utilization. However, with aperiodic/sporadic flows this efficiency is lost leading to longer response times and even unnecessary packet losses under peak loads due to the rigidity of the periodic schedules.

Therefore, similarly to the general case referred earlier, we need adequate flow management to achieve low response time and deterministic peak performance with differentiated quality of service (QoS) levels while improving

[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.

average bandwidth utilization. A scheduling paradigm that is particularly well suited to support this flow management is that of servers [2]. This paradigm has been recently applied to traffic scheduling in Ethernet switches particularly using an FTT-enabled Ethernet Switch [3] [4]. In this paper, the authors further extend such work analyzing the mechanism that allows adapting and reconfiguring the servers on-line while maintaining the negotiated QoS levels of each flow.

The paper is organized as follows. Section 2 discusses two particularly related frameworks, those of TTEthernet and PROFINET-IRT, while the FTT-enabled Ethernet Switch is briefly described in Section 3, with the implementation of the servers being addressed in Section 4. Sections 5 and 6 present the admission control algorithm and the adaptation and reconfiguration protocol, respectively, while Section 7 concludes the paper and points to lines of future work.

2 Related Work

Ethernet is probably the most pervasive networking protocol, having found application in many different domains, including the embedded systems domain where real-time constraints abound. Specifically to support real-time communication on Ethernet, several approaches were proposed. A first class of approaches, such as traffic shapers [5] or master-slave protocols [6], builds upon standard Ethernet switches. Usually, these solutions require a specific layer in the nodes network stack for accessing the real-time services. On the other hand, most of these protocols assume that their nodes are always well behaved and do not allow the connection of legacy nodes, ensuring thus the robustness of the system and the integrity of the real-time services.

On the other hand, another class of approaches uses modified switches. TTEthernet [7] is a new scalable real-time Ethernet platform that enables the seamless communication for distributed applications. It aims at supporting communication among applications with various real-time and safety requirements (Time-Triggered, Rate-Constrained, Best-Effort) over network. It is used for safety-critical, fail-operational applications and audio/video systems. The scheduling for the Time-Triggered traffic is performed off-line by a planning tool ensuring interference-free transmissions. Moreover, each end node implements traffic shapers to handle the Rate-Constrained class. Schedulability of this traffic can be assessed based on the rates and buffers sizes, which must be known a priori.

Another protocol relying on customized Ethernet switches is PROFINET - IRT [8]. Its switches offer deterministic transmission through explicit bandwidth reservation for the real-time data. The scheduling parameters are configured during the setup phase and they are obtained with a scheduling algorithm that is executed off-line. On the other hand, this switch also allows the integration of Ethernet standard devices whose traffic is confined in dedicated time windows that do not interfere with the periodic traffic.

In what concerns handling traffic that exhibits burstiness and jitter, the latter class of protocols offers a much stronger solution since the traffic confinement is carried out directly by the network devices, independently of the end nodes. However, there is no bandwidth reclaiming and if a flow is causing a transient overload in its slot, it cannot use the bandwidth of other slots that might be underloaded at that point. Moreover, since the definition of the communication slots is fixed, coping with structural changes in the application is strongly limited, for example when a given flow needs a durable modification in the bandwidth assigned to it, or a flow is added/removed from the set.

The solution presented in this paper targets supporting those cases in a bandwidth efficient manner by using server-based scheduling inside the switch and providing a structure that allows modifying the servers on-line.

3 FTT-enabled Ethernet Switch

The FTT-enabled Ethernet Switch was created from the Flexible Time-Triggered (FTT) paradigm [3] where the FTT master was inserted inside the switch (Master Module in Figure 1). The FTT protocol defines three traffic classes: 1) periodic real-time messages activated by the master (referred to as *synchronous* since their transmission is synchronized with the periodic traffic scheduler); 2) aperiodic or sporadic real-time traffic, autonomously activated by the application within each node; and 3) non real-time traffic (classes 2 and 3 are referred to as *asynchronous*). The synchronous and asynchronous traffic are transmitted within separate windows with the former typically having priority over the latter. The non real-time traffic is scheduled in background, within the asynchronous window. Moreover, the part of the synchronous window that is not used by synchronous traffic is made available for the asynchronous one, for the sake of bandwidth efficiency.

The referred windows alternate within fixed Elementary Cycles (ECs), Figure 2. Each EC starts with one master poll message, called Trigger Message (TM). The TM contains the synchronous schedule for that particular EC and the master never schedules more messages in an EC than those that fit in the respective synchronous window, thus memory overflows inside the switch are completely avoided for such kind of traffic. On-line changes in the synchronous message flows go through an admission control inside the master and are taken into account by the master on-line scheduler as soon as they are committed.

The asynchronous traffic is managed through queues inside the FTT-enabled Ethernet Switch and transmitted in

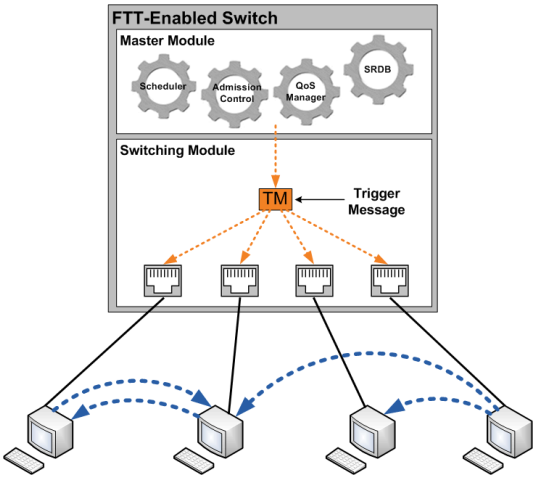

Figure 1. FTT-enabled Ethernet Switch.

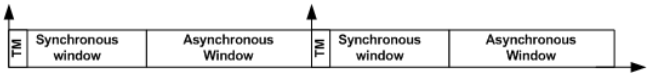

Figure 2. Elementary Cycles.

the asynchronous windows (Figure 2). In the case of the asynchronous real-time traffic, the switch regulates it by enforcing a minimum inter-transmission time per flow. This allows carrying out schedulability analysis of this kind of traffic within an on-line admission control, similarly to the case of the synchronous one.

Summarizing, the FTT-enabled Ethernet Switch was designed to provide the following advantages:

1. online admission control, dynamic quality-of-service management and arbitrary traffic scheduling policies;
2. an increase in the system integrity with unauthorized real-time messages being blocked at the switch input ports;
3. the asynchronous traffic is autonomously triggered by the nodes with arbitrary arrival patterns;
4. a standard node can take advantages of the real-time services through a simple negotiation with the switch, or without negotiation, the nodes can transmit as non-real-time traffic in a best-effort way and using all bandwidth left free by the synchronous and asynchronous real-time traffic and without interfering with it;
5. it can be configured in a fully synchronous mode, just with synchronous window, in a mixed mode using both windows in which it is possible to configure the maximum width of the synchronous window and fully asynchronous mode, i.e., with just asynchronous window.

4 Servers in the FTT-enabled Ethernet Switch

In recent work we proposed using the scheduling flexibility of the FTT-enabled Ethernet Switch to carry out server-based traffic scheduling [4] in which case the asynchronous flows, either real-time or non-real-time, are handled by servers.

Two ways were considered for adding server-based scheduling to the FTT-enabled Ethernet Switch. One solution implements the admission control algorithm, the servers structures and the scheduler inside the Master Module, while the servers queues are left in the Switching Module. The Master Module then periodically scans the servers queues and updates the servers structures, directly implementing the servers policy and instructing the Switching Module of which packets of which servers should be transmitted in each EC. This solution allows implementing any desired server policy and is very scalable in the number of servers that can be supported at the cost of a penalty in latency due to the sequence of operations involved between the reception of a packet in a server and its transmission.

Conversely, we also pursued another solution in which the servers structures and scheduler are implemented in hardware, within the Switching Module, just leaving the admission control in software in the Master Module. This way, the Master Module is invoked solely when a new server is created, deleted or updated. Apart from these situations, server packets are completely handled in hardware with very low latency. The negative aspect of this option is a reduction in the flexibility concerning server policies and scheduling because of limitations in their hardware implementation. Moreover, the hardware resources needed to implement the servers structures must be allocated from the beginning and are limited, limiting the maximum number of servers that can be used.

Both solutions were implemented but their actual comparison is out of the scope of this paper, which focuses on the functionality needed to manage the servers creation, deletion and adaptation at run-time, particularly the admission control and its associated schedulability analysis.

5 Admission Control

In order to make sure that the set of servers can meet their timing requirements, i.e., that they can get their capacity within their replenishment period when needed, an adequate admission control is invoked before creating a new server or updating its temporal properties. Moreover, a second test is also performed upon server creation or adaptation in order to test whether its memory requirements can be met.

Currently, we will consider sporadic servers, only, due to their better performance in schedulability and latency.

5.1 Traffic and Servers Model

The server-based scheduling is implemented within the asynchronous window with minimum duration LAW per EC (duration E). In the case of a fully asynchronous implementation, i.e., without synchronous window and even without the TM, the asynchronous window occupies the whole cycle, with $LAW = E$. However, in the general case, $LAW = E - LTM - LSW$, where LTM is the time needed to transmit the TM and LSW is the maximum duration of the synchronous window per EC.

The asynchronous streams are characterized as in (1) where N_{as} is their total number, C_i is the maximum transmission time of stream i, $Tmit_i$ represents the respective minimum interarrival time, Srv_i is the server to which stream i is allocated and s_i its sender node. Finally, DL_i is the set $\left\{dl_i^1, .., dl_i^{k_i}\right\}$ of k_i destination links for stream i.

$$AS_i = (C_i, Tmit_i, Srv_i, s_i, DL_i), DL_i = \left\{dl_i^1, .., dl_i^{k_i}\right\}, i = 1..N_{as}. \quad (1)$$

On the other hand, the servers are characterized as in (2), where N_{srv} is the current number of servers in use. The individual server Srv_j is characterized by its capacity C_j, its replenishment period T_j, and a few figures extracted from the set $\left\{AS_1^j, .., AS_{m_j}^j\right\}$ of m_j asynchronous streams allocated to it, namely the maximum and minimum transmission times $Cmax_j$ and $Cmin_j$ respectively, and the set of destination links $SDL_j = \bigcup_{c=1}^{m_j} DL_c^j$.

$$Srv_j = (C_j, T_j, Cmax_j, Cmin_j, SDL_j), j = 1..N_{srv} \quad (2)$$

Finally, note that for each destination link dl_p with $p = 1..N_{dl}$ (the number of destination links) the respective asynchronous window will be shared by the set t_p of active servers.

$$t_p = \left\{Srv_1^p, ..Srv_{t_p}^p\right\}, p = 1..N_{dl} \quad (3)$$

5.2 Schedulability Analysis

As we have seen in the model description above, the server Srv_j will potentially execute in a number of destination (output) links (Figure 3). However, in all links where it executes, it keeps its capacity and period. The actual messages that a server transmits in each output link at each instant in time can differ, though.

Therefore, we say that a server is schedulable if and only if it is schedulable in the set SDL_j of destination links in which it is active. The whole system is schedulable when all servers are schedulable or, equivalently, when all destination links are schedulable.

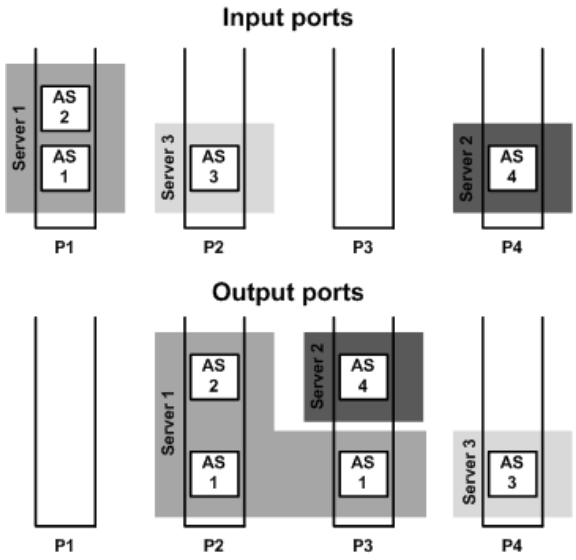

Figure 3. Servers in the input and output links.

For the sake of simplicity of implementation and speed of execution, we propose a schedulability test based on the utilization of the destination links, or more specifically their asynchronous windows. Moreover, it is important to recall that the basic transmission control mechanism does not allow a packet in the asynchronous window to start transmission if it cannot terminate within the window. This corresponds to inserting idle-time at the end of the asynchronous window and it is called the blocking-free non-preemptive scheduling model. This model allows avoiding the blocking that otherwise the asynchronous packets would cause to the TM, if present, and it was proposed and thoroughly analyzed in [9], from where we extract the following theorem:

Theorem 1 [9]: Any exiting schedulability analysis for fixed priorities preemptive scheduling can be used with the *blocking-free non-preemptive model* if the tasks execution times (C_i) are inflated by a factor $E/(E - X)$, where E is the EC duration and X the maximum inserted idle time.

Moreover, we make use of another previous result concerning the schedulability analysis of servers, in particularly sporadic servers [10], from where we reproduce the following theorem:

Theorem 2 [10]: A periodic task set with fixed priorities that is schedulable with a task τ_i is also schedulable if τ_i is replaced by a sporadic server with the same period and execution time.

Hence, from a schedulability point of view, analyzing a set of sporadic servers is equivalent to analyzing the schedu-

lability of a set of corresponding tasks. By putting together both theorems, we can analyze the schedulability of one particular destination link dl_p running a set t_p of sporadic servers as if it was a set of fixed priority preemptive tasks with a capacity inflated by a factor equal to $E/(E-X)$. In this case, X represents the total inserted idle time in each cycle from the point of view of the asynchronous window, including the time to transmit the TM, the synchronous window and the actual inserted idle-time in the asynchronous window itself. X can then be upper bounded by $E - LTM - LSW - Cmax^p = LAW - Cmax^p$, where $Cmax^p = max_{j=0}^{t_p}(Cmax_j)$. The inflated server capacities in the output link dl_p are expressed as in 4.

$$C'^p_j = E/(LAW - Cmax^p) * C_j, j = 1...t_p \quad (4)$$

Note that this result allows using any schedulability analysis for fixed priorities preemptive scheduling. In particular, we use the Liu and Layland's utilization bound for Rate-Monotonic Scheduling with just the small adaptation of the capacities as referred in 4 and use it within a fast on-line admission control for a set of active servers in a given destination link (dl_p) as follows:

$$U(dl_p) = \sum_{j=0}^{t_p} \left(\frac{C_j}{T_j}\right) < N_s \left(2^{1/t_p} - 1\right) \left(\frac{LAW - Cmax^p}{E}\right) \Rightarrow \quad (5)$$

the set of t_p active servers in dl_p is schedulable with RM under any phasing.

5.3 Memory Analysis

The FTT-enabled Ethernet Switch currently implements a simple but deterministic memory management based on blocks of fixed size equal to the longest Ethernet frame. Therefore, server Srv_j in the worst case requires $MemBlk_j$ memory blocks, given by expression 6. This amount of memory must be available at the time of creation or adaptation of a server to have the respective request accepted.

$$MemBlk_j = \left\lceil \frac{C_j}{Cmin_j} \right\rceil \quad (6)$$

6 Adaptation and Reconfiguration Protocol

In order to operate an FFT-enabled Ethernet Switch with server-based scheduling it is necessary to be able to create, delete and update servers as well as to associate streams to servers. These operations are carried out inside the switch and are requested by the application using special purpose FTT messages (FTT Request messages). In order to allocate streams to servers, all streams must be uniquely identified. However, note that the switch can be simultaneously used by FTT-compliant and FTT-agnostic nodes, including for the real-time services provided with the servers, as described in Figure 4. Therefore, the stream identification for each case uses different parameters. The streams from FTT-compliant nodes are encapsulated according to the FTT protocol and identified by the Ethernet MAC destination and source addresses and by the FTT stream identifier. On the other hand, the streams transmitted by FTT-agnostic nodes and encapsulated with the Internet Protocol version 4 are identified based on the IP source and destination addresses, source and destination ports, and transport protocol.

Moreover, note that the requests for setting up the servers and streams of an FTT-agnostic node can be made from a third party node. Once the proper configuration is done, the FTT-agnostic node can communicate through the respective link and server(s).

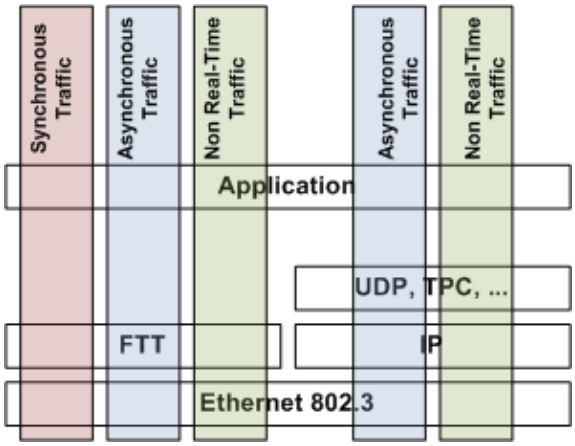

Figure 4. Communication model.

Each FTT Request triggers the admission control algorithm, being the result communicated back to application node through an FTT Reply message. Five types of FTT Request messages (Figure 5) are defined:

- reserve a new server to manage a set of streams
- add new streams to a server
- remove streams from a server
- update the server parameters (capacity and period)
- release a running server

Each of these messages has an associated FTT Reply message (Figure 6).

When an FTT Request message arrives at the switch, the Switching Module identifies it (Figure 7) and sends it

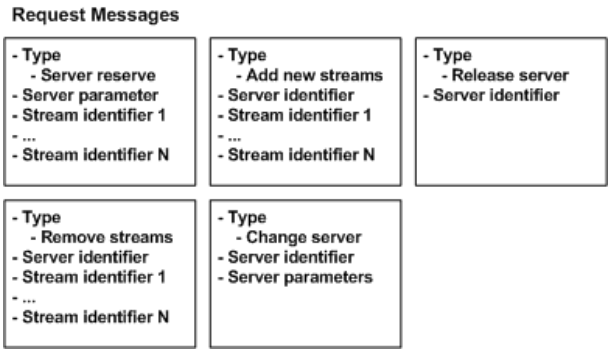

Figure 5. Request Messages.

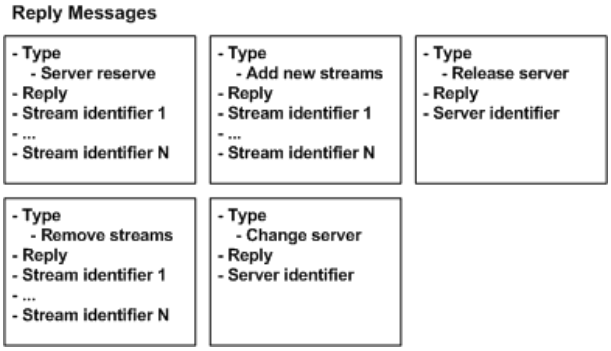

Figure 6. Reply Messages.

promptly to the Master Module. The requests are stored within a FIFO queue while the Admission Control executes one by one, at the beginning of each EC. The response is communicated to the Switching Module that builds an FTT Reply message and sends it back to the source node of the request during the asynchronous window of the following EC. Note that this mechanism allows just one reply per EC, thus bounding the impact of possible bursts of requests. This reply message per EC is also accounted for in the schedulability analysis of the servers in the asynchronous windows, which basically corresponds to using shorter windows.

One relevant parameter is the reactivity of this protocol with respect to the requests. The time that it takes since a request arrives at the switch to the time at which the respective reply is sent back to the requester (RTD) is shown in 7, where N_RP is the number of requests pending inside the Master Module. Bounding RTD requires a careful analysis of the application to also bound N_RP.

$$RTD = E + N_RP * E + E + LTM + LSW \qquad (7)$$

After a successful admission control, the servers structures and parameters are configured. This time, however, is neglectable when compared the RTD latency.

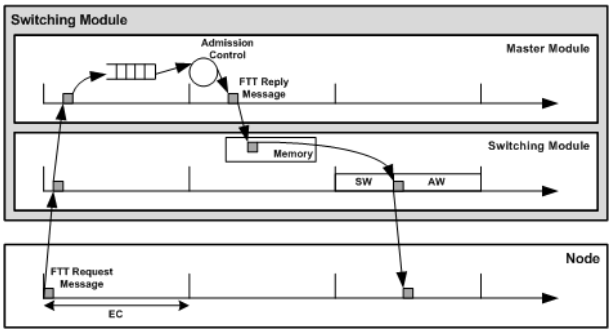

Figure 7. Adaptation and reconfiguration process.

7 Conclusion

In order to face the growing connectivity requirements, with more abundant and diverse information to be exchanged, it is important to support arbitrary arrival patterns in an efficient way. In this paper we focused on real-time Ethernet networks, based on switches, to support such trend. We realized that existing protocols are particularly efficient with synchronous/periodic communications with slot-based and static approaches, becoming inefficient with aysnchronous/aperidic traffic that is characterized by arbitrary arrival patterns.

On the other hand, a network based on flow management is recognized as an effective means to achieve low response time and provide differentiated QoS levels improving average bandwidth utilization. Therefore, in order to overcome the limitations above, the authors recently proposed integrating the server-based scheduling within an FTT-enabled Ethernet Switch. The resulting framework provides efficient servers implementation, namely sporadic servers, online admission control and dynamic quality-of-service management of the servers.

In this paper the authors further extended the work of integrating the server-based scheduling paradigm in an FTT-enabled Ethernet Switch. The paper presents a solution for the admission control algorithm considering both timeliness, using previous results and based on bandwidth utilization, and memory usage. Finally, the reconfiguration and adaptation protocol is briefly described, and an upper bound on the reaction time to change requests in the servers/streams structure is shown. This protocol is still under development and further assessment will be carried out in future work, including an assessment of the overhead and efficiency of the proposed admission control and its effective integration with dynamic QoS management.

References

[1] L. Roberts, "The Internet is broken. Let's fix it." *IEEE Spectrum*, July 2009.

[2] C. Buttazzo, *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series)*. Santa Clara, CA, USA: Springer-Verlag TELOS, 2004.

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

[4] 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 the 2nd Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS'09) in conjunction with the 30th IEEE International Real-Time Systems Symposium (RTSS'09)*. Washington DC - USA: , December 2009.

[5] J. Loeser and H. Haertig, "Low-Latency Hard Real-Time Communication over Switched Ethernet," in *ECRTS '04: Proceedings of the 16th Euromicro Conference on Real-Time Systems*, Catania, Sicily, Italy, 2004, pp. 13–22.

[6] R. Marau, P. Pedreiras, and L. Almeida, "Enhancing Real-Time Communication over COTS Ethernet Switches," in *WFCS 06 - The 6th IEEE Workshop on Factory Communication Systems*. Turin - Italy: IEEE Computer Society, June 2006.

[7] W. Steiner, G. Bauer, B. Hall, M. Paulitsch, and S. Varadarajan, "TTEthernet Dataflow Concept," in *NCA '09: Proceedings of the 2009 Eighth IEEE International Symposium on Network Computing and Applications*. Cambridge, MA USA: IEEE Computer Society, 2009, pp. 319–322.

[8] PROFInet, "Real-Time PROFInet IRT," http://www.profibus.com/pn, December 2007.

[9] L. Almeida and J. Fonseca, "Analysis of a Simple Model for Non-Preemptive Blocking-Free Scheduling," in *ECRTS '01: Proceedings of the 13th Euromicro Conference on Real-Time Systems*. Delft, The Netherlands: IEEE Computer Society, 2001, p. 233.

[10] B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic task scheduling for hard-real-time systems." *Journal of Real-Time Systems*, 1989.