Flexible, Efficient and Robust Real-Time Communication with Server-based Ethernet Switching *

R. Santos, A. Vieira, P. Pedreiras, A. Oliveira
DETI / IEETA / University of Aveiro
Aveiro, Portugal
{rsantos,alexandrevieira,pbrp,arnaldo}@ua.pt

L. Almeida, R. Marau
IEETA - DEEC / University of Porto
4200-465 Porto, Portugal
{lda,marau}@fe.up.pt

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

Abstract

The information exchanged in Networked Embedded Systems is steadily increasing in quantity, size, complexity and heterogeneity, with growing requirements for arbitrary arrival patterns and guaranteed QoS. One of the networking protocols that is becoming more common in such systems is Ethernet and its real-time Ethernet variants. However, they hardly support all the referred requirements in an efficient manner since they either favour determinism or throughput, but not both. A potential solution recently proposed by the authors is the Server-SE protocol that uses servers to confine traffic associated to specific applications or subsystems. Such an approach is dynamically reconfigurable and adaptive, being more bandwidth efficient while providing composability in the time domain. This paper proposes integrating the servers inside the Ethernet switch, boosting both the flexibility and the robustness of Server-SE, allowing, for example, the seamless connection of any Ethernet node. The switch is an FTT-enabled Ethernet Switch and the paper discusses two specific ways of integrating the servers, namely in software or in hardware. These options are described and compared analytically and experimentally. The former favours flexibility in the servers design and management while the latter provides lower latency.

1. Introduction

There has been a continued steep increase in the complexity, quantity and heterogeneity of the data exchanged between nodes in Networked Embedded Systems (NES). From data originated in simple 10 bit ADCs to multi-kilobyte variable bit-rate multimedia traffic. Moreover, many NES are frequently subject to real-time constraints that extend to the respective information exchanges, requiring support from a real-time network. One network technology that became widely used in these systems is Ethernet [1], which conquered the office automation market long ago, entered massively into the factory automation and large embedded systems domains and is now being considered for mass market domains such as the automotive one. However, Ethernet was not originally developed to meet the requirements of NES, namely in what concerns key aspects such as predictability and timeliness. These limitations led to the development of the so-called Real-Time Ethernet (RTE) protocols, but even these still reveal difficulties in handling the variety of requirements that current NES pose in an efficient manner, particularly arbitrary arrival patterns and widely different QoS requirements. Typically, such protocols were either tuned to achieve high bandwidth efficiency or strict timeliness guarantees but not both.

Standard Ethernet switches are typically designed for high throughput Internet access or file sharing, presenting limitations in what concerns real-time performance, with potentially long queueing delays or even packet losses resulting from limited scheduling capabilities (up to 8 prioritized FIFO queues, only) and a generalized lack of memory partitioning. The techniques proposed to overcome such limitations range from shaping the traffic submitted to the switch [2] to limiting that traffic by application design [3], adding transmission control features [4] and providing more efficient scheduling policies and admission control [5] [6] [7]. Amongst the most proeminent market contenders we can find EtherCAT [4], PROFINET [6], Ethernet/IP [3] and TTEthernet [7].

One of the main challenges in designing current NES is managing the ever growing level of complexity [8] [9] [10]. Component-oriented design methodologies, which provide composability, are particularly well suited since they support safe resource sharing, allowing different components/subsystems to be developed separately and later integrated in the system. On the other hand, server-based scheduling is recognized as an effective means to enable such kind of resource sharing [11] and it can be the basis for resource partitioning and virtualization, supporting

*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.

978-1-4244-5461-7/10/$26.00 ©2010 IEEE

the separation between the applications software architecture and the hardware platform on which they will execute. Such separation has the potential to bring significant cost reductions at the system level and is currently the objective of active frameworks such as AUTOSAR [12] in the automotive domain, IMA [13] in avionics or IEC61499 [14] in industrial automation.

However, the support for network partitions by current RTE protocols suffers from limitations imposed by specific medium access control and queue management policies within network devices and protocol stacks that do not support efficient server-based traffic scheduling policies. Moreover, 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. Additionally, the respect for network partitions is frequently delegated to the end nodes, which 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 access control layers.

To address some of these limitations, the authors previously proposed the Server-SE protocol [15], building on top of FTT-SE [16] to support arbitrary server scheduling policies and servers hierarchical composition in a dynamically but deterministic reconfigurable way. More recently, the authors proposed integrating server scheduling techniques inside an FTT-enabled Ethernet Switch [17]. The resulting platform preserves the properties of Server-SE, particularly the support for heterogeneous traffic classes (periodic, aperiodic; real-time, non-real-time), arbitrary (server) scheduling policies and dynamic adaptation and reconfiguration. In addition, operating at layer 2 allows providing highly efficient and robust network partitions that cope with arbitrary traffic arrival patterns, thus allowing, for example, the seamless connection of general purpose nodes, which have no specific adaptation, without affecting the QoS guarantees provided by the user-defined servers that reside in the switch. A proof-of-concept implementation was described in [18].

In this paper we explore two possible options for integrating the servers scheduling unit inside the FTT-enabled Ethernet Switch architecture, namely in software, in the FTT Master Module, or in hardware, in the Switching Module. This design option has important consequences in terms of responsiveness, flexibility, hardware complexity and global system schedulability. This paper presents both architectural solutions, analyzes them and compares them with respect to the referred parameters, both analytically and experimentally using prototype implementations.

The remainder of the paper is organized as follows: Section 2 presents an overview of related work; Section 3 presents a brief introduction to FTT-SE and the FTT-enabled Switch; Section 4 describes the server-based traffic scheduling implementation in software and hardware and discusses their advantages and disadvantages; Section 5 presents a prototype implementation of both architectures as well as experimental results and, finally, Section 6 presents the conclusions.

2. Related work

2.1. Introduction to servers

The server mechanisms have been introduced in the context of CPU scheduling as a means to handle hybrid task sets composed by periodic and aperiodic tasks, with the objective of limiting the interference that each aperiodic task can have on other tasks (both periodic or aperiodic) while providing minimum levels of service to the aperiodic tasks. On the other hand, a server can be defined as an entity that controls the access to a given resource by a task or set of tasks. The server is the entity that is scheduled and aperiodic tasks are executed on its behalf. Thus, by giving appropriate levels of priority and service to the servers, the access to the resources managed by them can be strictly controlled.

Many server types are described in the literature, both for fixed and dynamic priority systems. Amongst the most well known fixed-priority servers we can find the Polling Server (PS), the Deferrable Server (DS) and the Sporadic Server (SS) [19]. The PS and the DS operate in a similar way, being characterized by a period and a capacity, also known as budget, that is replenished every period. They differ when no aperiodic tasks are pending, with the PS suspending itself until the beginning of the next period while the DS preserves its capacity and can execute until the end of the period. For this reason, the DS has a better average response time to aperiodic requests but at the cost of a reduced system schedulability. The SS can be regarded as an improvement over both the PS and DS since it can execute at any time, if there is capacity available (as the DS), but without penalizing the system schedulability. To illustrate the server concept, Figure 1 shows the operation of a PS (s) with intermediate priority between two periodic tasks (1 and 3), replenishment period $T_s = 4$ and capacity $C_s = 1$ time units. When there are outstanding aperiodic requests, the next server instance executes as if it was another periodic task.

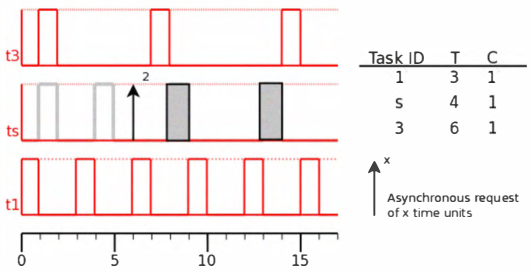

Figure 1. Example of Polling Server

For the case of dynamic priority systems there are adapted versions of the servers referred above but also specific ones, such as the Constant Bandwidth Server (CBS) [20] and the Total Bandwidth Server (TBS) [21]. This case is, however, out of the scope of this paper and will not be detailed here (see [22] for further information).

Finally, another kind of typical server is the Background Server (BS) that becomes active for execution whenever there is nothing else to be executed in the system, thus corresponding to the lowest priority level.

The work reported in this paper uses Sporadic Servers. These can execute at any time, if capacity is available, while enforcing the bandwidth corresponding to the (C_s, T_s) pair. This requires a specific budget replenishment rule that is no longer strictly periodic, as for the PS and the DS, being instead scheduled whenever the capacity is consumed:

- Being RT the replenishment time and T_s the server period, if the server becomes active at t_A and remaining $capacity > 0$ then $RT = t_A + T_s$.
- The replenishment capacity RA to be done at time RT is computed when the sporadic server becomes idle or C_s has been exhausted. Let t_I be such a time. The value RA is set equal to the capacity consumed within the interval $[t_A, t_I]$.

In the worst-case, when there are always outstanding aperiodic requests, the SS exhibits a periodic behavior for all purposes equivalent to that of a periodic task with the same period and execution time equivalent to the server budget, which is particularly useful for schedulability analysis.

2.2. Servers in the Ethernet network domain

The terminology associated to servers in the networking domain is frequently different from that used in CPU scheduling. For example, a common server used in networking is the *leaky bucket*. This is a specific kind of a server category called *traffic shapers* [23], 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 of 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. However, it is hard to categorize these network servers similarly to the CPU servers because networks seldom use clear fixed or dynamic priority traffic management schemes. In fact, there is a large variability of Medium Access Control (MAC) protocols, some of them mixing different schemes such as round-robin scheduling, first-come-first-served, multiple priority queues, etc.

Focusing now on RTE protocols, some limited forms of server-based traffic handling can be found. PROFINET RT and IRT [6] present bi-phase periodic communication cycles, comprising a mandatory Real-Time (RT) phase optionally followed by a non-RT (NRT) phase. The RT schedule is built off-line and downloaded to the switch at configuration time. The protocol depends on a custom switch to enforce the cyclic structure and traffic confinement. The protocol operation can be regarded as a polling server, devoted to the periodic traffic, composed with a background server, dedicated to the NRT traffic.

TTEthernet [24] is also based on a customized switch that enforces a TDMA framework. When there is no RT traffic, nodes can transmit arbitrary NRT data. Whenever a TDMA slot is scheduled, the switch aborts current ongoing NRT transmissions, if any, making sure that the communication medium is free for the RT transfer. The underlying TDMA framework allows the existence of event slots thus, globally, the operation of this protocol can be regarded as a set of polling servers (off-line scheduled event slots) combined with a background server that handles the NRT traffic.

Ethernet Powerlink [25] implements a cyclic dual-phase communication structure, with one phase devoted to the isochronous traffic and the other to aperiodic traffic. The protocol operation can be regarded as the composition of two polling servers, one devoted to the isochronous traffic and the other to the asynchronous traffic. Within the isochronous window the messages are served cyclically, thus the protocol behaves as a polling server. The access control is carried out via a Master-Slave scheme. The periodic real-time traffic requirements are supposed to be known at system design time and the corresponding schedule is built offline and downloaded to the network master node. Aperiodic traffic is served on demand, and messages are sorted according to a fixed priority scheme.

The switch architecture proposed by Wang *et al* [26] is based on a clock-driven scheduler that serves each input queue in a cyclic manner, approaching a polling server. The traffic is supposed to be static and known during the system design. The port scheduling matrix is then computed offline and downloaded to the switch, at pre-runtime.

Finally, other protocols, such as [23], implement traffic shapers in the end nodes, managed by suitable software modules, which behave similarly to a DS.

Due to infrastructural limitations, none of these protocols supports arbitrary server policies nor their hierarchical composition and dynamic adaptation or creation/removal, features that are provided by the FTT-enabled Ethernet switch described in this work.

3. FTT-enabled Ethernet Switch

The FTT-enabled Ethernet Switch is a modified Ethernet switch based on the Flexible Time-Triggered (FTT) paradigm and arouse from the work carried out in the scope of the FTT-SE protocol.

3.1. FTT-SE and Server-SE

FTT-SE [16] is a real-time Ethernet protocol for micro-segmented switched Ethernet networks with COTS switches that follows the Flexible Time-Triggered (FTT) paradigm. It uses a master/multi-slave transmission control technique in which a master addresses several slaves with a single poll that is broadcast once every so-called Elementary Cycle (EC) of fixed duration using a specific message called Trigger Message (TM). This system supports synchronous (periodic) traffic, with periods that are integer multiples of the EC, asynchronous (aperiodic) traffic,

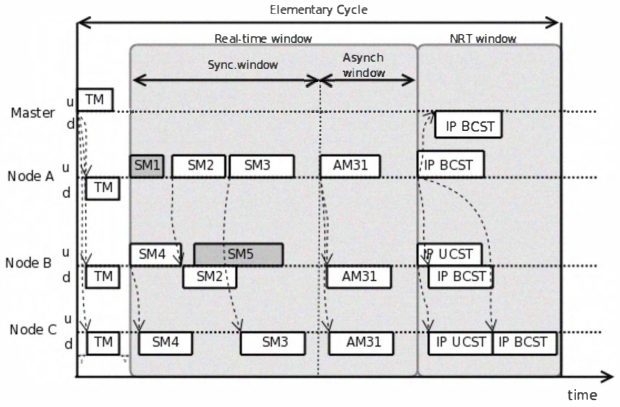

Figure 2. Traffic scheduling in FTT-SE.

which is handled with lower priority than the synchronous one, and non-real-time (NRT) traffic that is handled in the background, using the time left in each EC, after the synchronous and asynchronous traffic (Figure 2).

The synchronous traffic is scheduled on-line by the master and the respective schedules per EC are disseminated with the TM (Figure 2). Nodes decode the TM and transmit immediately the scheduled messages with the switch taking care of their serialization. The scheduling is done so that all messages referred in a TM fit in the respective EC. Thus, message queuing in the switch is strongly bounded and queues are always flushed by the end of each EC. The FTT master holds information about the nature of the data exchanges regarding the type of addressing (unicast, multicast and broadcast) and which end nodes are involved. With this information the master computes which messages follow disjoint paths (i.e., non overlapping source and destination nodes) and thus build schedules that exploit this parallelism, increasing the aggregated throughput.

The strict confinement of the asynchronous traffic, required for real-time operation, is achieved in FTT-SE by extending the transmission control role of the master node also to this traffic class. The slave nodes report, at the beginning of each EC, the status of their asynchronous queues to the master, which then schedules the asynchronous transmission requests, via the TM, at the appropriate instants and in strict observation of the EC duration and structure. This mechanism resembles the approach used in other real-time protocols, e.g. Ethernet PowerLink and WorldFIP. It is, however, substantially more bandwidth efficient since there is no need for the master to periodically poll the slaves. FTT-SE takes advantage of the full-duplex links so that slaves report the status of their aperiodic traffic queues to the master approximately at the same time that the master is sending the TM. The asynchronous signaling mechanism in presented in detail in [27].

Server-SE [15] is a specialization of the FTT-SE protocol configured for asynchronous traffic, only, and in which all the traffic is handled by servers. These can handle a single stream each or a set of messages related to one application or node and can be composed hierarchically. Server-SE is thus an important piece in supporting composability in the time domain in a distributed system being capable of handling arbitrary arrival patterns while maintaining timeliness guarantees.

3.2. FTT-Enabled Ethernet Switch

However, FTT-SE, and consequently Server-SE, present some structural limitations that cannot be solved with standard Ethernet switches. These arise from the fact that the protocol assumes all nodes to be FTT-compliant, i.e., respecting the EC schedules transmitted in the TM and the respective timings. Therefore, a specific network device driver is needed, which might not be available in several operating systems, and misbehaving nodes not respecting the protocol timings will jeopardize all timing guarantees. The solution to overcome these problems is to add traffic confinement capabilities, particularly temporal control, to the switch. This was achieved inserting the FTT master inside the switch. The switch generates the TM in each EC, thus polling the synchronous traffic directly. Conversely, the asynchronous traffic need not be polled, being queued inside the switch in dedicated memory pools, and transmitted when appropriate, e.g., within servers, or simply enforcing a minimum inter-transmission time. Thus, the switch basically shapes the traffic in the outgoing links so that it conforms to desired (negotiated) timing patterns.

Summarizing, the FTT-enabled Ethernet Switch was designed to provide the following features, some of which are simply inherited from FTT-SE:

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.

4. Server scheduling integration architectures

The integration of server-based traffic scheduling in the FTT-enabled Ethernet Switch can be carried either in software, within the Master Module, or in hardware, inside the Switching Module (Figure 3). This design option results in differentiated behaviors in terms of responsiveness, flexibility, hardware complexity and global system schedulability.

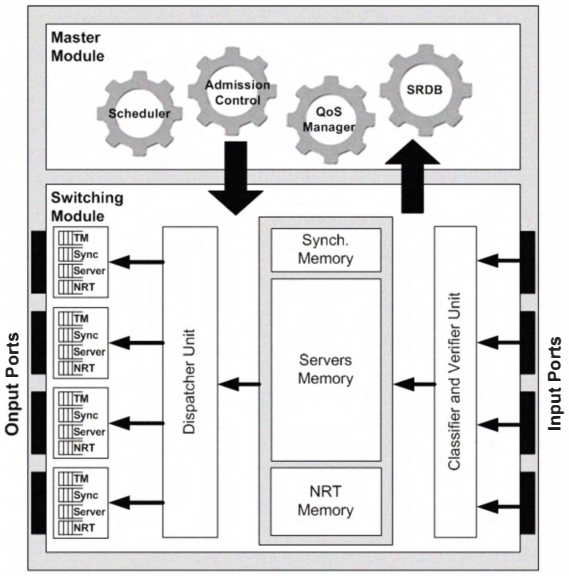

Figure 3. Switch functional architecture.

This section explores these two architectural design options, showing their operation principles and presenting a qualitative comparison among them.

4.1. Server-based traffic scheduling implemented in software

In the software implementation the servers are managed by the Master Module. From the logical operation point of view, this approach is essentially equivalent to the Server-SE protocol [15]. The servers are, in this case, software entities that are accepted, created, modified, deleted and scheduled in the Master Module.

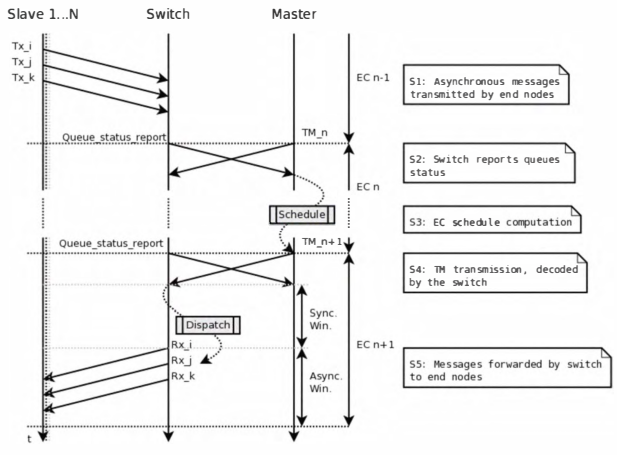

Figure 4. Software implementation event sequence

The management of the servers is carried out upon application request via standard FTT configuration messages (FTT Requests) that are passed on to the Master Module. Similarly to FTT-SE, the result of the negotiation procedure is reported back to the requester entity. However, in this case, it is also intercepted by the Switching Module logic, which uses the report result to self-configure the necessary resources, namely memory blocks and the table with the streams allowed for each server. The memory blocks are specifically tailored for each server, guaranteeing the absence of memory conflicts among different servers.

After this initial set-up phase the servers become fully functional and enter in normal operation mode (Figure 4). The nodes may then issue messages at any instant, which are intercepted and stored by the Switching Module logic (step S1). At the end of every EC the Switching Module reports the queues status to the Master Module (step S2). Then, the Master builds the schedule for the following EC, which takes into account all the pending requests (step S3). The EC schedule is then communicated to the Switching Module, via the TM, at the beginning of the following EC (step S4). The Switching Module intercepts the TM and, finally, forwards the previously queued messages to the destination nodes when the asynchronous window comes (step S5).

Figure 4 shows that the minimum delay a message may experience happens when it arrives at the switch at the end of the EC, just before the instant in which the Switching Module reports the queues status to the Master Module. In this scenario the server is scheduled with a delay of one EC (*LEC*) (recall that the Master Module always schedules the traffic one EC in advance). Then, the message has to wait for the Trigger Message (*LTM*) and the EC Synchronous Window (*LSW*) duration before being transmitted. Also note that in store-and-forward mode, as in the current prototype, the transmission time C_i must be accounted for twice. Finally, the switching logic also imposes a small delay, represented by *SD*. Therefore, the minimum latency that an asynchronous message i, with transmission time C_i, may experience ($L^{sw}_{min_i}$) is given by Equation 1.

$$L^{sw}_{min_i} = LEC + LTM + LSW + 2 * C_i + SD \qquad (1)$$

In the absence of contention with other asynchronous messages, one asynchronous message experiences a worst-case latency ($L^{sw}_{max_i}$) when it arrives at the switch in the instant immediately after the Switching Module reporting its queues status to the Master Module. Since the report only occurs once in each EC, an additional delay of one *LEC* is added. Thus, the value of $L^{sw}_{max_i}$ is given by Equation 2.

$$L^{sw}_{max_i} = 2 * LEC + LTM + LSW + 2 * C_i + SD \qquad (2)$$

This simple analysis highlights the impact of the explicit signalling mechanism with a period of one EC to post the scheduler about the current server requests. The relatively and potentially large duration of the EC leads, thus, to a loss of reactivity.

From a logical point of view this scheme exhibits essentially the same properties as the Server-SE protocol, supporting arbitrary server scheduling policies as well as their composition, combined with a tight integration with the flexible Master Module scheduling, admission control and QoS management. However, contrarily to Server-SE, non-compliant or misbehaving nodes no longer can jeopardize the system timeliness because all the incoming traffic is screened, subject to sanity checks and, if necessary,

trashed. Additionally, the access to the servers is transparent to the end nodes, allowing the connection of nodes that do not implement the FTT-SE protocol while, at the same time, providing them with guaranteed QoS levels. This feature is particularly valuable to incorporate legacy nodes or to manage the system complexity, by providing transparent network partitions that are managed completely within the network.

4.2. Server-based traffic scheduling implemented in hardware

The low reactivity of the software-based architecture can be improved partitioning the servers between software and hardware. The configuration tasks, related mainly with admission control, schedulability analysis and QoS negotiation, are complex, hard to implement in hardware and relatively tolerant to delays, thus they are left in software, within the Master Module. On the other hand, the tasks related with the online server management and message forwarding are supported in hardware, guaranteeing low latency and high predictability. In this approach the servers are created at synthesis time and consequently their type and number are not changeable dynamically. Every time a new server is added, removed or its parameters modified, the negotiation results are sent to the Switching Module, which reconfigures the servers operational parameters accordingly. A more dynamic architecture, permitting the dynamic creation and removal of servers, would require online FPGA reconfiguration and is beyond the scope of this paper.

In this approach the server scheduling is carried out autonomously by the Switching Module, independently of the scheduler in the Master Module. Thus, two schedulers co-exist, the Master Module scheduler handling the synchronous traffic in the synchronous window and the servers scheduler handling the asynchronous and non-real-time traffic in the asynchronous window. Specifically, during the asynchronous window the hardware scheduler/dispatcher unit verifies continuously the status of each server FIFO queue and, following the scheduler rules, forwards immediately all messages that fit in that window. Reactivity is thus improved with respect to the software-based architecture.

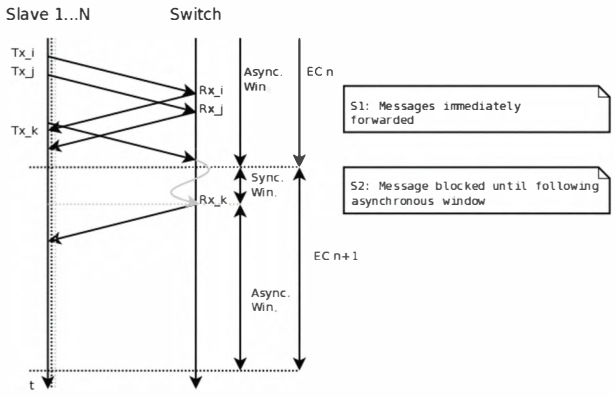

Figure 5. Hardware implementation event sequence

Figure 5 shows that in the best case scenario the latency is essentially the message transmission time plus the switching latency, a situation that happens whenever a transmission fits within the current asynchronous window (scenario S1). In the absence of contention from other asynchronous messages, the worst-case latency happens when the message is sent to the switch near the end of the asynchronous window and does not fit in for a small amount. In that case, to prevent window overruns the switch will not forward any message that arrives within a configurable guarding window (equivalent to the transmission time of the longest possible asynchronous message). Thus, the value of $L^{hw}_{max_i}$ is given by Equation 3, where LGW represents the guarding window duration.

$$L^{hw}_{max_i} = LGW + LTM + LSW + 2 * C_i + SD \qquad (3)$$

Therefore, the hardware-based implementation presents higher reactivity that is independent of the EC duration. Additionally, for simple scheduling algorithms such as Rate Monotonic or Round-Robin, the implementation is resource-efficient and fairly simple, requiring basically one counter, one timer and some logic per server.

However, the hardware-based architecture compares negatively with the software one in terms of flexibility. The maximum number and type of servers has to be defined at pre-runtime, despite their allocation, deallocation and reconfiguration being done at runtime. On the other hand, complex server scheduling methods can require a significant amount of hardware resources. Furthermore, from the average bandwidth efficiency point of view, the software architecture can also outperform the hardware one since, at scheduling time, the size of the messages waiting in the switch queues are known, which allows optimizing the use of the asynchronous window.

5. Experimental results

This section presents experimental results extracted from two prototype implementations, one of each architecture, which allow assessing the correctness of the servers operation in terms of bandwidth guarantees, traffic isolation and latency bounds.

Both implementations are based on a 4 port FTT-enabled Ethernet Switch architecture, following a similar Hw/Sw co-design approach as proposed in [28]. The prototype switch implements the Switching Module in hardware using a NetFPGA board [29], integrating a Virtex-II Pro XC2VP50 FPGA. The Master Module is implemented in software, running in an independent CPU, connected to the FPGA via a dedicated Ethernet link on *Port 4*.

A common scenario is used to allow comparisons, consisting of an EC with a duration of 1ms with the asynchronous window using 42% of the EC. There are two sporadic servers, SS1 and SS2, with a budget of 3200B(Bytes) and period 1ms each, plus a background server BS that reclaims the bandwidth left free by the sporadic servers.

Figure 6 describes the setup used in the experiments. A video stream is simultaneously fed through servers SS1

and BS, while a time-bounded constant load, simulating an UDP transaction, is fed to SS2. The video stream was analyzed offline, offering an average load of around $10Mbps$, with peaks that reach $21.9Mbps$. Additionally, when the video application is launched it sends a burst of messages that lasts for $77ms$ and offers an average load of $87.4Mbps$. For the sake of efficiency of the channel bandwidth utilization, this period is considered as an initialization phase, during which no real-time guarantees are given. Note that this burst is completely dependent on the specific streaming application. For applications requiring shorter initialization phases it would be possible to tune the streaming software to use a lower amount of buffers, thus reducing the initial burst length. The load fed through SS2 is active from the instant $t = 23s$ to $61s$ and, when active, generates a constant load of $58.6Mbps$. A simple assessment of the load bandwidth submitted to the servers allows concluding that when the video streams experience peak activity the bandwidth is insufficient, leading to overloads. SS1 has the highest priority and thus the video stream served by it should not be degraded during overloads. The simulated UDP traffic is served by the lower priority SS2. Since the bandwidth allocated to SS1 and SS2 exceeds the asynchronous window capacity, during peak activity on SS1, SS2 may also not be able to receive the full bandwidth. Finally, the video stream fed through the BS is expected to experience a severe quality degradation when SS2 is active, since the BS has no guaranteed bandwidth.

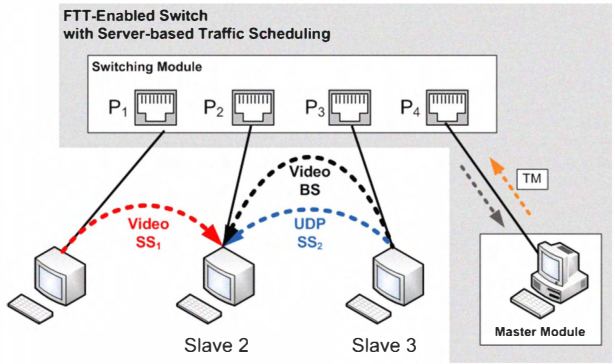

Figure 6. Setup

5.1. Software implementation

The implementation of the servers in software, inside the Master Module, requires for the hardware components of the switch 49% of the board FPGA total slices, allowing a maximum operation frequency of $127.13MHz$.

Figure 7 shows, for each server, the difference between the submitted traffic at the input ports of the FTT-enabled Ethernet Switch and the allowed traffic at output ports. The first graph shows the submitted traffic (video stream) to server SS1 and the corresponding allowed traffic. The two curves overlap, meaning that the traffic managed by this highest priority server is forwarded without significant losses and without a noticeable delay. The second graph represents their difference which, as expected, is essentially null. The third graph shows the submitted and allowed traffic (video stream) managed by BS. Between the instants $t = 23s$ to $61s$, i.e., when the traffic submitted to SS2 is active, there is a significant deviation between the submitted traffic and the allowed traffic. This means that the bandwidth left free by servers SS1 and SS2 is not enough for transmitting all the traffic submitted to the BS, eventually leading to packet losses. On the other hand, there are cases in which the submitted traffic is below the allowed (outgoing) traffic, a situation that occurs due to buffering in the switch when the bandwidth allocated to the server is enough to flush the buffered messages. The fourth graph shows the difference between the submitted and allowed traffic for the BS. A significant amount of dropped packets occurs while SS2 is active. Finally, the last graph shows the UDP simulated load submitted to SS2. This graphs shows the traffic isolation provided by the servers. SS2 confines its output traffic to the allocated capacity, trashing the incoming packets in excess of its buffering capacity. There are also fluctuations in the allowed traffic of SS2 caused by peaks of bandwidth use by SS1 that has more priority.

Table 1 shows the total number of packets transmitted and effectively forwarded by the switch during the experiment, for each server. The numeric results confirm the observations from Figure 7. The highest priority server SS1 experiences a marginal packet loss while servers SS2 and BS experience higher packet losses, particularly BS due to its lowest priority. Note that SS1 only loses packets during the initialization phase, when the average submitted load largely exceeds its allocated bandwidth, as mentioned before. After $t = 77ms$ no further packet losses occur. This phenomenon can be observed in the second graph of Figure 7, where it is visible that packets are dropped only at the beginning of the experiment.

	SS1	SS2	BS
Submitted Packets	57854	183778	57328
Forwarded Packets	57398	72529	33877

Table 1. Number of packets submitted and allowed (software implementation).

To assess the reactivity of the software-based architecture we used the configuration described previously, with a packet generator, served by the SS1, sending periodically 1500B packets ($C_i = 122\mu s$) to the switch, with equal MAC destination and source addresses, causing the switch to return the packet to the sender. The packets are generated with a period equal to one EC plus a small offset, $16ns$, to cause packet arrivals at the switch in every different relative phase of the EC structure. The round-trip delay was measured at the generator node, computing the time difference between the instant in which each packet starts to be transmitted by the node and the corresponding instant in which the same packet starts to be received back in the sender node. For consistency, this requires subtracting C_i to the latency equations in Section 4.

Moreover, the value of SD was experimentally determined to be $2.8\mu s$. The other relevant timing parameters

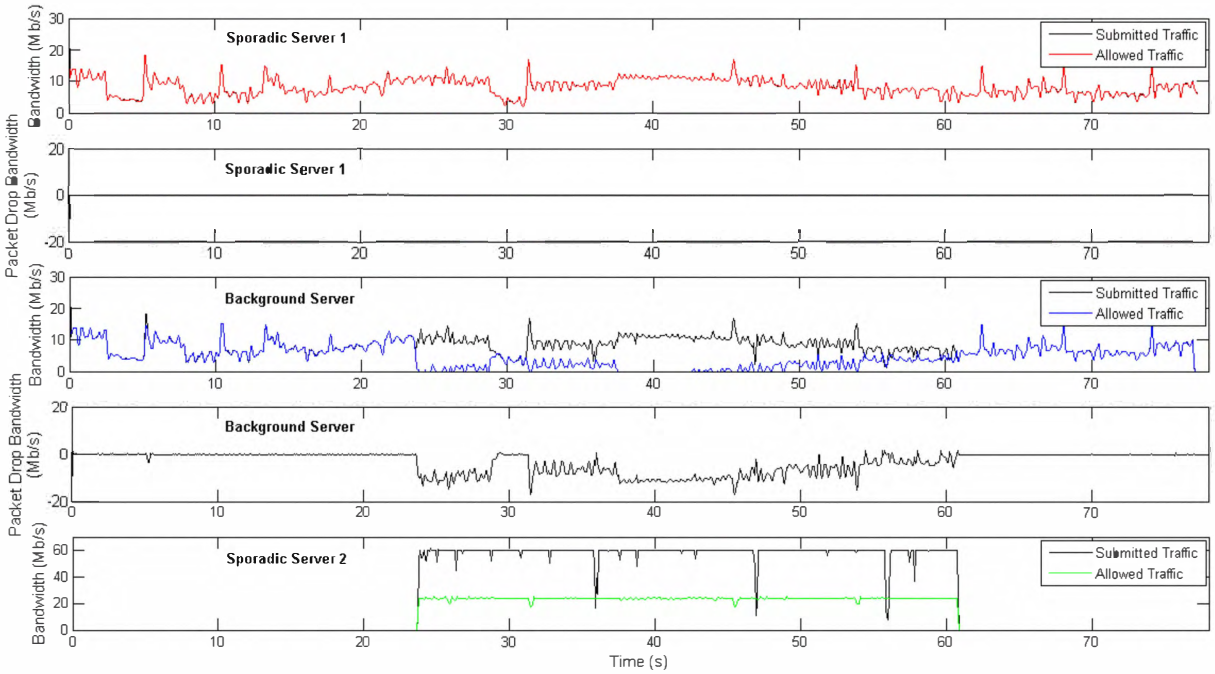

Figure 7. Submitted and allowed load difference in software implementation.

associated with the EC configuration are shown in Figure 9.

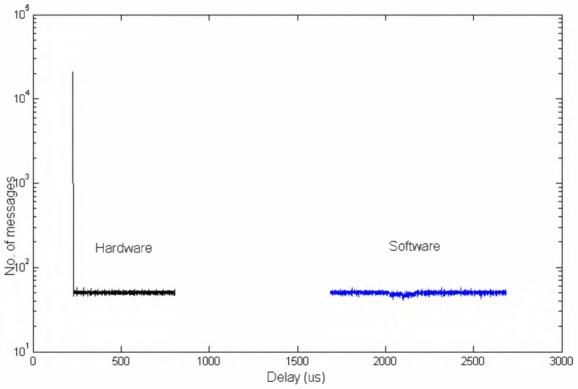

Figure 8. Histogram of the minimum server latency distribution of both architectures.

Figure 8 presents the histogram of the results which exhibit a strong match with the expected ones. The estimated best and worst-case round-trip delays obtained from Equations 1 and 2 are $L^{sw}_{min_i} = 1684.78\mu s$ and $L^{sw}_{max_i} = 2684.78\mu s$ while the actual measured round-trip delays were comprised between $1684,8\mu s$ and $2684.8\mu s$.

5.2. Hardware implementation

As opposed to the software implementation, the hardware implementation requires an amount of FPGA resources that depends on the maximum number of servers. To characterize this dependency we instantiated a growing

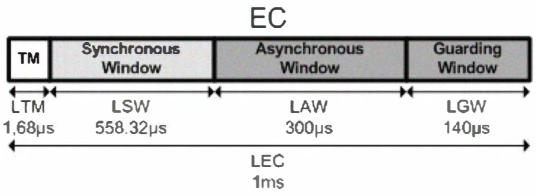

Figure 9. EC implementation.

number of servers until reaching the full FPGA utilization. The results are show in Figure 10, with a linear dependency between the number of servers and the amount of resources required. The maximum operating frequencies are also presented. We consider that frequencies remain constant, however, the small registered variations are due to the optimization algorithm in the synthesize application.

A set of tests regarding the servers bandwidth utilization, similar to the one carried out for the software implementation was also carried out. As expected, the graphs obtained for the diverse servers are essentially equal to the ones presented in Figure 7, since a similar configuration was used, and thus are not repeated.

The total number of packets transmitted and effectively forwarded by the switch in this case are reported in Table 2. As the configuration was similar for both implementations, these results are essentially similar to those reported in Table 1, as expected.

Concerning the reactivity of the hardware implemented servers we measured the round-trip delay similarly to the software case and as referred in Section 4.2. This delay is expected to be comprised between $125.8\mu s$, case in which

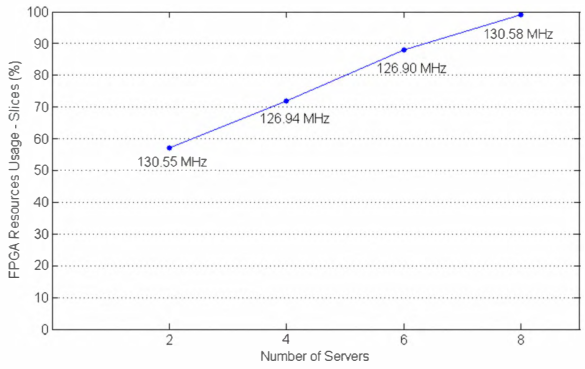

Figure 10. Number of server vs. used resources.

the message forwarding fits within the asynchronous window, and $L^{hw}_{max_i} = 825.8\mu s$ as given by Equation 3, otherwise. The measured delay varied between $125.8\mu s$ and $826\mu s$ again showing a strong consistency with the respective estimates (Figure 8). Moreover, there is also a high number of occurrences of the lower bound delay ($125.8\mu s$), corresponding to the situation in which the packets are received by the switch and can be forwarded during the asynchronous window.

6. Conclusions

Served-based traffic scheduling is an efficient way of supporting resource partitioning which is considered an important building block to provide composability and allow tackling the growing complexity and heterogeneity of current Networked Embedded Systems. One fundamental resource in such systems is the network and enforcing network partitions that provide temporal isolation with timeliness guarantees and arbitrary arrival patterns is a desired objective. To reach this objective the authors recently proposed the Server-SE protocol that applies server-based traffic scheduling over COTS Ethernet switches. This approach, however, suffers from the limitations of such switches and, thus, the authors also proposed a new switch, the FTT-enabled switch.

In this paper, the authors proposed integrating the servers inside the new switch, enhancing the flexibility and robustness of the Server-SE solution, without jeopardizing the timeliness guarantees. Particularly, the paper explores two options for such integration, with the servers implemented in software or hardware, and compares them in terms of server responsiveness, flexibility, hardware complexity and global system schedulability. In global terms, the software implementation, using the Master Module,

	SS1	SS2	BS
Submitted packets	58473	181337	57447
Forwarded packets	58014	73498	23980

Table 2. Number of packets submitted and allowed (hardware implementation).

provides greater flexibility in creating, managing and deleting servers. Moreover, it allows elaborated global scheduling, managing different kinds of traffic in an integrated way and is more scalable. On the other hand, the hardware implementation, using the Switching Module, offers greater reactivity, an important aspect in order to maintain the advantages of some servers. On the other hand, it allows a limited number of servers, only, and simple scheduling algorithms.

The paper presented a prototype implementation of both architectures and their experimental assessment. The results show the feasibility and correctness of the approaches that were considered. We believe that the FTT-enabled switch enhanced with server-based scheduling presented in this paper offers an effective and efficient support to network partitions, boosting the applicability of switched Ethernet in complex NES, as required by mainstream frameworks such as AUTOSAR, IMA or IEC61499.

References

[1] J-D. Decotignie. Ethernet-based real-time and industrial communications. *In Proceedings of the IEEE*, June 2005.

[2] J. Loeser and H. Haertig. Using Switched Ethernet for Hard Real-Time Communication. In *Proceedings of IEEE International Conference on Parallel Computing in Electrical Engineering*, September 2004.

[3] Open DeviceNet Vendors Association. Ethernet/IP. http://www.odva.org/.

[4] EtherCAT Technology Group. EtherCAT - Ethernet for Control Automation Technology. http://www.ethercat.org, December 2007.

[5] S. Varadarajan and T. Chiueh. EtheReal: A Host-Transparent Real-Time Fast Ethernet Switch. In *Proceedings of International Conference on Network Protocols*, October 1998.

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

[7] M. Plankensteiner. TTEthernet enabes the use of Ethernet networks in all applications. *Embedded Control Europe*, pages 12–14, 2008.

[8] K. Grimm. Software technology in an automotive company - major challenges. In *Proceedings of IEEE International Conference on Software Engineering*, May 2003.

[9] A. Phillips, D. Yanakiev, and F. Jiang. Managing complexity in large-scale control system design. In *Proceedings of the American Control Conference, 2004*, June 2004.

[10] B. Rumpler. Complexity Management for Composable Real-Time Systems. In *Proceedings of IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, 2006.

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

[12] Autosar. http://www.autosar.org/, accessed 10/Nov/2009.

[13] J. Littlefield-Lawwill and R. Viswanathan. Advancing Open Standards in Integrated Modular Avionics: An Industry Analysis. In *Proceedings of IEEE International Digital Avionics Systems Conference*, October 2007.

[14] IEC 61499 homepage. http://www.holobloc.com, accessed 10/Nov/2009.

[15] R. Marau, N. Figueiredo, R. Santos, P. Pedreiras,

L. Almeida, and T. Nolte. Server-based Real-Time Communications on Switched Ethernet. In *Proceedings of International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, December 2008.

[16] R. Marau, P. Pedreiras, and L. Almeida. Enhancing Real-Time Communication over COTS Ethernet Switches. In *Proceedings of IEEE Workshop on Factory Communication Systems*, June 2006.

[17] 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 IEEE Workshop on Factory Communication Systems*, May 2008.

[18] 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 International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, December 2009.

[19] L. Sha B. Sprunt and J. Lehoczky. Aperiodic Task Scheduling for Hard Real-Time Systems. In *Real-Time Systems*, 1989.

[20] L. Abeni and G. Buttazzo. Integrating Multimedia Applications in Hard Real-Time Systems. In *Proceedings of IEEE International Real-Time Systems Symposium*, December 1998.

[21] M. Spuri and G. C. Buttazzo. Efficient Aperiodic Service under Earliest Deadline Scheduling. In *Proceedings of IEEE International Real-Time Systems Symposium*, December 1994.

[22] G. Buttazzo. *Hard Real-Time Computing Systems - Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, 1997.

[23] 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.

[24] TTTech. TTEthernet. http://www.tttech.com/solutions/ttethernet/, November 2008.

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

[26] 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.

[27] R. Marau, P. Pedreiras, and L. Almeida. Asynchronous Traffic Signaling over Master-Slave Switched Ethernet Protocols. http://rtn2007.loria.fr/, jul 2007. RTN 2007, 6th Workshop on Real-Time Networks, Pisa, Italy.

[28] 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 IEEE Industrial Electronics Society*, November 2009.

[29] NetFPGA. http://www.netfpga.org/, May 2009.