Implementing Server-Based Communication within Ethernet Switches

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

Server-based architectures have generated recently a considerable interest. They provide an effective means to support composability, i.e., the integration of diverse components while guaranteeing the required service-levels to each one. While common in CPU scheduling, the support for server-oriented architectures in the domain of real-time communication protocols is more limited due to distribution and specific medium access control and queues management policies within network controllers, network devices and protocol stacks. Consequently, server-based traffic scheduling is either not supported or supported in a limited and inefficient way, e.g., only basic servers, no hierarchical composition, static configuration. To overcome such limitations, the authors proposed recently the Server-SE protocol, which supports unconstrained server-based traffic scheduling over switched Ethernet, using the FTT-SE protocol and common off-the-shelf (COTS) switches as platform. This paper extends such work by bringing the servers inside a customized Ethernet switch. This option provides a high level of determinism, robustness and flexibility, being particularly suited to open systems as servers can easily be added, composed, adapted and removed at run-time. The proposal is validated with a prototype implementation and experimental results that show its effectiveness in enforcing correct resource reservations.

1 Introduction

The development of complex embedded systems greatly benefits from composability, i.e., the capability of integrating diverse components in a system in which they share resources while satisfying their individual service requirements and enforcing mutual temporal isolation. Server-oriented architectures are 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 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 object of active frameworks such as AUTOSAR in the automotive domain, IMA in avionics or IEC61499 in industrial automation. These frameworks are generally distributed and thus require an appropriate support to distributed partitions that naturally include network segments.

The current support for network partitions suffers from limitations imposed by specific medium access control and queues management policies within network controllers, network devices and protocol stacks that do not allow efficient server-based scheduling policies as those developed for CPU scheduling. 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. Finally, the respect for network partitions is frequently delegated to the end nodes that must execute a specific layer on top of the general network interface, typically a traffic shaper, which is a limitation for the integration of legacy systems and other general purpose systems that do not originally include such layer. Moreover, even in the cases in which such layer can be effectively deployed, the proper operation of the system requires the compliance of all system components to a correct temporal behavior.

Therefore, in order to support network partitions that are not limited by the underlying protocol or network interfaces/devices the authors proposed previously the Server-SE protocol [2], integrating the FTT-SE [3] and Server-

CAN [4] protocols, the former providing a master/slave architecture that supports operational flexibility and the latter providing an integrated server-based traffic scheduling paradigm. Server-SE provides a seamless integration of real-time and non-real-time services, with strict timeliness guarantees to the first class. Arbitrary server scheduling policies are supported including their hierarchical composition. Furthermore, the servers properties can be changed dynamically, e.g., to deal with changes in the application requirements or environment, without compromising the timeliness of the real-time services. However, the implementation over FTT-SE still requires that a specific software layer be installed above the network device driver in all nodes to control transmissions adequately.

Recently, the FTT-SE framework was complemented with a costumized Ethernet switch [5] that integrates the FTT master functionality and is capable of traffic classification and policing at the input ports. This latter feature allows confining the incoming traffic to reconfigurable time windows, whichever its type and arrival pattern. This capability is not present in current real-time Ethernet (RTE) protocols and is particularly well suited for supporting open distributed real-time systems. Note that legacy or general purpose systems can now be connected to the network without requiring any specific software adaptation layer and their traffic is transparently confined to windows that are disjoint with respect to other sources of time-sensitive traffic, providing mutual temporal isolation.

In this work, the authors propose exploiting the new FTT-enhanced switch to deploy the Server-SE protocol, supporting the servers directly on the switch hardware. By doing this, the protocol is now able to provide network partitions, or virtual channels, that are flexible with respect to their hierarchical composition, dynamic adaptation and reconfiguration and server scheduling policy while being robust to arbitrary traffic arrival patterns.

These features are a significant contribution to the field of RTE protocols, which are extensively used in complex embedded and/or industrial systems, where composability frameworks are particularly useful. Therefore the paper is organized as follows: Section 2 presents a brief discussion about the server-class services supported by some relevant RTE protocols; Section 3 revisits the basics of the FTT-Enabled Switch and its FPGA-based implementation; Section 4 presents the core of the proposal, detailing the integration of server-based traffic scheduling into the switch; Section 5 presents a prototype implementation, showing the plausibility of the proposal and its capability to guaranteeing the correct server temporal behavior, even in the presence of interference with arbitrary arrival patterns and load variations. Finally, Section 6 presents the conclusions.

2 Server-based traffic scheduling

In the networking domain, probably for historical reasons, the names given to servers are different from those used in CPU scheduling. For example, a common server used in networking is the *leaky bucket*. This is a specific kind of a general server category called *traffic shapers* [6], which purpose is to limit the amount of traffic that a node can submit to the network within a given time window, bounding the node burstiness. These servers use techniques similar to those used by CPU servers, based on capacity that is eventually replenished. Many different replenishment policies are also possible, being the periodic replenishment as with the Polling Server (PS) or the Deferrable Server (DS), the most common ones. 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.

Particularly regarding RTE protocols, some very limited forms of server-based traffic handling can be found. Some protocols enforce periodic communication cycles with reserved windows for different traffic classes (e.g. PROFINET-IRT [7], TTEthernet [8] and Ethernet Powerlink [9]). This is a trivial composition of several PS that hardly supports an efficient use of the network bandwidth. Other protocols, such as [6], 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. These features are provided by the Server-SE implementation proposed in this work, and are inherited from the new FTT-enabled Ethernet switch.

3 FTT-enabled Ethernet switch

3.1 Brief overview

The FTT-enabled switch is based on the Flexible Time-Triggered (FTT) paradigm with the FTT master included 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 the background, within the asynchronous window. For the synchronous traffic, a master/multi-slave transmission control technique is used, according to which a master addresses several slaves with a single poll message, considerably alleviating the protocol overhead when compared to the conventional master-slave techniques. The communication is organized in fixed duration slots called Elementary Cycles (ECs). Each EC starts with one poll message sent by the master, called Trigger Message (TM). The TM contains the schedule for that particular EC. Only the messages that fit within an EC are scheduled by the master, thus memory overflows inside the switch are completely avoided for such kind of traffic.

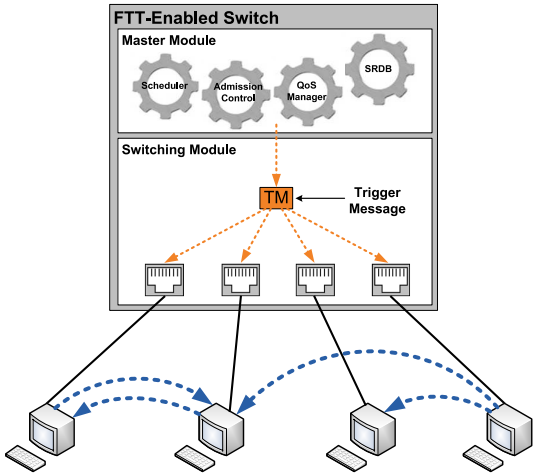

Figure 1. FTT-enabled Ethernet switch

Integrating the FTT master in the switch preserves most FTT attributes while obtaining important gains in the following key aspects:

- A simplification in handing the asynchronous traffic that is now autonomously triggered by the nodes instead of being polled by the master node, while maintaining aggregated or per-stream temporal isolation;
- An increase in the system integrity since unauthorized real-time transmissions can be readily blocked at the switch input ports, thus not interfering with the rest of the system.
- Seamless integration of non-FTT-compliant nodes without jeopardizing the real-time services.

3.2 Switch architecture

The functional architecture of the FTT-enabled Ethernet switch has been presented and discussed in earlier work [5]. It is basically formed by four main blocks, the *master* itself, which includes the System Requirements Data Base, the admission controller, the synchronous scheduler and Quality of Service (QoS) manager, the *input blocks* that classify, validate and filter the ingress traffic, the *global memory pool* that holds the messages of each class in independent sections, and the *output blocks* that include three pointer queues, one for each traffic class, and assure the jitter-free transmission of the TM in each EC.

This architecture allows us maintaining a tight control on the traffic that enters the switch, including enforcing an adequate timing behavior and temporal isolation among traffic classes, whichever is the traffic arrival pattern. Therefore, nodes producing NRT or asynchronous traffic can use the switch transparently, as if it was a standard Ethernet switch, without needing any modification of the node software. This is particularly relevant to cope with legacy applications that were not designed to use the synchronous services. The former traffic will not interfere with the latter and both will not interfere with the synchronous traffic that might be flowing through the switch. This grants a high flexibility to the proposed solution for real-time communication, which efficiently combines those heterogeneous traffic classes with mutual temporal isolation. An appropriate FTT network driver is just required for the synchronous communication services and for setting up asynchronous channels dynamically.

4 Integrating server-based scheduling in the FTT-enabled switch

In [2] the authors addressed the integration of server-based traffic scheduling with the FTT-SE protocol and proposed Server-SE. Despite supporting arbitrary servers, as well as their hierarchical composition and dynamic adaptation, creation and removal, the Server-SE protocol is based on COTS Ethernet switches and thus depends on the exclusive presence of FTT-compliant nodes in the network segment, otherwise the real-time properties are compromised. This limitation is now overcome by using the FTT-enabled switch to support Server-SE, exploiting its capability to classify, confine and validate traffic at the switch ingress ports. The protocol then provides the server hierarchy depicted in Figure 2.

At the top level the FTT EC structure uses two disjoint windows (SW and AW), that fill in the whole Elementary Cycle (EC), to handle the two main traffic classes, i.e., synchronous and asynchronous. These windows appear once in each EC (period E) and have a bounded size (LSW and LAW, respectively), specified in the FTT configuration. Several deployment alternatives can be considered. Typically, the synchronous window is a polling server with period $T_{SW} = E$ and capacity $C_{SW} = LSW$ while the asynchronous window is a deferrable server with lower priority

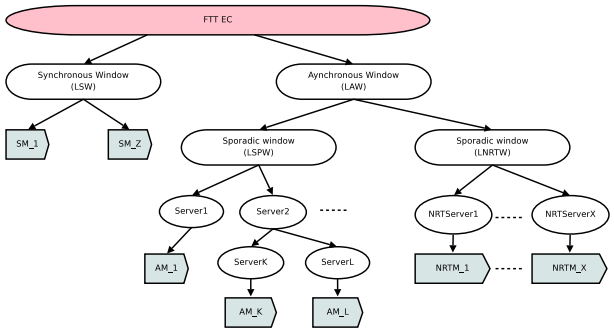

Figure 2. Hierarchy of servers

than the former but reclaiming the space it leaves unused in each EC. Its period is $T_{AW} = E$ and its capacity C_{AW} is inheritied from the EC after having removed the capacity of the synchronous server and other protocol overheads leading to a minimum of $C_{AW} = LAW = E - LSW - \delta$. A less efficient alternative but simpler to implement, and the one used in the prototype described later on, is to deploy both windows as two polling servers with fixed capacity, scheduled in a TDMA fashion. The bandwidth of the synchronous server is $U_{SW} = \frac{C_{SW}}{T_{SW}} = \frac{LSW}{E}$ and of the asynchronous one $U_{AW} = \frac{C_{AW}}{T_{AW}} = \frac{E-LSW-\delta}{E} = 1 - U_{\delta} - U_{SW}$. Note that E and LSW, and also δ to some extent, are FTT configuration parameters that can be tuned to suit the global application needs in terms of synchronous and asynchronous requirements. Particularly, LSW can take any value from 0 to $E - \delta$ controlling the bandwidth distribution between the two servers. In general, a lower LSW smoothes the synchronous load across ECs and improves the responsiveness to asynchronous requests.

The second level of the hierarchy manages the sporadic and NRT traffic, the former having real-time requirements and thus being always scheduled before the latter that is handled with a background server. Thus, at this level, the sporadic window inherits the capacity and bandwidth of its parent server ($C_{SPW} = C_{AW}; U_{SPW} = U_{AW}$) while the NRT server inherits the remaining capacity left free in the EC.

The third level of the hierarchy is where additional application-specific servers can be plugged-in, constituting virtual channels. These servers can be implemented with arbitrary scheduling policies without preemption at the packet level. The sole constraints are that the base time granularity for periods, deadlines and offsets is E and their aggregated bandwidth cannot exceed U_{SPW}.

4.1 Provided services

The server allocation follows a similar procedure to the one defined for the Server-SE implementation. All nodes have to negotiate with the switch the creation of adequate servers in order to handle specific types of traffic. The node requests are issued via specific FTT control messages, while the TM, which is sent by the switch, conveys the request replies. The negotiation of the server parameters, i.e, the parameters of the desired virtual channel, is based on admissible ranges of QoS. The switch implements an admission control module assuring that, at any time, the switch has enough resources to satisfy the real-time requirements of the traffic that is conveyed within the negotiated channels. During the communication process, the nodes can renegotiate the QoS parameters of any channel using the same service as for setting up channels. When a node stops using a channel it should delete the associated server, freeing up the communication resources, which eventually may be assigned to other active channels or to accommodate new ones.

The traffic classification and temporal confinement is carried out, transparently, by the switch. Therefore, legacy applications can seamlessly communicate through a virtual channel with QoS guarantees, provided that the channel is properly set-up. This may be achieved either by a third-party entity, e.g. another process in the same or any other node, or by manual switch pre-configuration. Additionally, legacy applications can also seamlessly communicate using the NRT background server implemented in each node. This server is created by default and does not require any kind of negotiation. The background server appears to the applications as one normal Ethernet link, with the exception that it presents a reduced and variable bandwidth since it inherits the capacity left free by the other servers. Thus it does not interfere with the QoS guarantees of the real-time channels, independently of the number of users it has.

4.2 Proposed functional architecture

Figure 3 presents the updated functional architecture of the switch to support server-based scheduling. It follows closely the one in [5] with two main modules, the Switching Module and the Master Module. The traffic arrives via the input ports in the former module and is submitted to the Classifier and Verifier Unit that classifies and validates the received massages. The data messages are forwarded directly to the memory unit while FTT control messages, e.g., negotiation messages, are transferred to the Master Module. The memory is divided in three independent zones, each one for each traffic class, namely synchronous, server and non-real time. The other main block inside the Switching Module is the Dispatcher Unit that handles the output queues per traffic type and, according to the scheduling performed by the master and conveyed in the Trigger Message, transmits the selected messages from the memory directly. The Master Module executes a complex set of operations,

namely the admission control, QoS manager, scheduler and it also implements a System Requirements Database to store the information related to the traffic management.

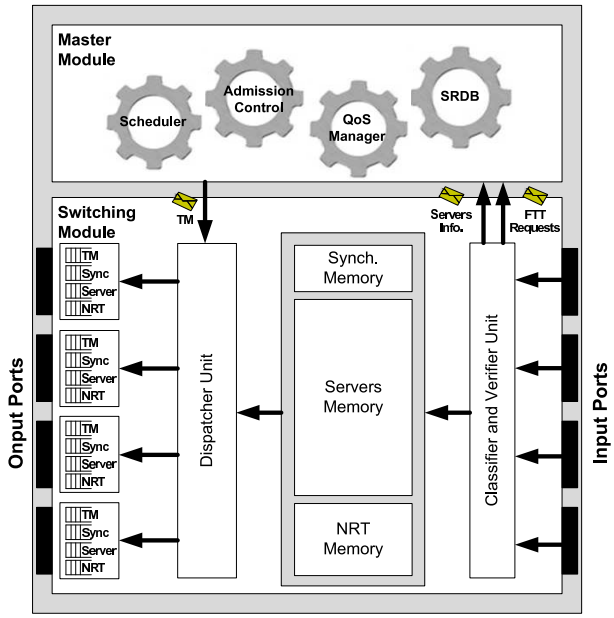

Figure 3. Switch functional architecture

The integration of server-based mechanisms in the FTT-enabled switch is carried out by associating one server instance to each asynchronous stream using the stream ID. Such association is carried out upon a server creation, which occurs both in the Master and Switching Modules. In the former, the server creation requests arrive via FTT control messages (FTT Requests in Figure 3) and trigger a QoS negotiation that results in specific server parameters that are then communicated back to the application via the TM. In the Switching Module, a FIFO with a requested depth is allocated to the server from the Servers Memory by the Classifier Unit.

When a node transmits asynchronous messages, the Classifier and Verifier Unit reads the stream ID and directs them to the associated FIFO queue in the memory. If a node transmits more messages than negotiated during a short period of time, the corresponding FIFO will fill up. When its limit is reached, further incoming messages are trashed.

In order to schedule the servers adequately, the Switching Module informs the Master Module, at the beginning of each Elementary Cycle, about which messages have been received by the servers in the previous EC (Servers Info in Figure 3). With this information the Master knows how much of the servers capacity is requested, information that is subsequently used for the scheduling of the following EC. The result of the scheduling is then communicated back to the Switching Module, via the TM, where the Dispatcher Unit enforces the respective transmissions. This process is illustrated in Figure 4. The whole process incurs in a latency of at least two ECs. This latency is the penalty to pay for having the servers scheduled by the FTT Scheduler, inside the Master Module. An alternative would be to have the servers scheduled autonomously in the Switching Module (a preliminary trial was already considered in [10]). However, this approach would make their dynamic adaptation, e.g., in the scope of dynamic QoS management, more difficult. The assessment of the trade-offs and implications among the diverse integration possibilities is not a trivial issue and will be subject of future work.

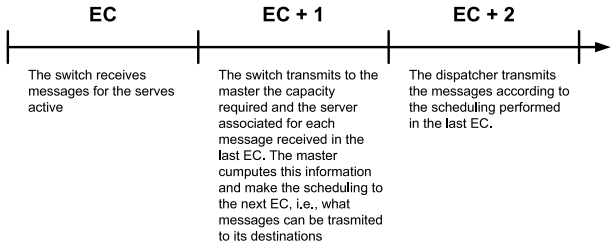

Figure 4. Servers forwarding process

When a communication channel is not needed anymore it can be closed upon explicit request. This operation frees the occupied resources, namely the FIFO in the Memory Unit and the control structures in the Master.

5 Experimental results

This section presents two different experimental setups, which address two distinct aspects of the switch-based Server-SE operation. The first experiment, based on a static scenario, shows the correct operation of both a sporadic server and a background server, namely the limitation of the bandwidth used by the associated message streams and the confinement of the asynchronous traffic to the asynchronous window. The second experiments is based on a dynamic scenario, in which the streams vary the submitted load at runtime. The purpose of this second experiment is to highlight the correctness of the hierarchical relation between the top-level polling server, which manages the asynchronous window, and the sporadic and background servers. Namely it is shown that the background server is able to dynamically reclaim the predecessors server bandwidth not used by the sporadic server.

The experimental results are obtained from a prototype implementation of the server-enabled Ethernet switch architecture following a similar Hw/Sw co-design approach as proposed in [10]. The prototype switch implements the Switching Module in hardware using a NetFPGA board [11], integrating a Virtex-II Pro XC2VP50 FPGA and using 42% of the board total slices, with a maximum operation

frequency of 127.13MHz. The Master Module is implemented in software, running in an independent CPU, a PC, connected to the FPGA by a dedicated Ethernet link on *Port 4*.

The first experiment integrates two traffic types in order to validate the switch traffic classification and confinement, as well as the bandwidth limitation according to the servers capacities and priorities. Figure 5 illustrates the setup. The Master Module is configured with an elementary cycle of 1ms, 29% of which assigned to the synchronous window, 54% to the asynchronous window and 16% for the guarding window. The remaining 1% is taken by protocol overheads, e.g., the TM transmission. The guarding window is a period of time at the end of the EC during which no new transmissions are allowed to start, in order to prevent EC overruns and so assuring that the TM transmission never suffers blocking. The switch is configured to full-duplex 100Mbit/s operation.

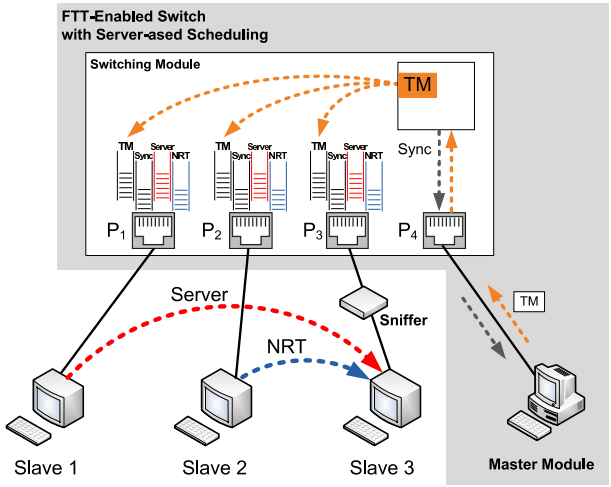

Figure 5. Experimental setup

In this scenario one node, *Slave 1*, sends 1500B size real-time messages to *Slave 3*, separated only by the Ethernet minimum inter-frame gap (96 bit times), thus generating a load close to 100% of the respective uplink ($M1$). An associated sporadic server, with a 3000B capacity and a period equal to two ECs handles this stream. Simultaneously, a third node, *Slave 2*, continuously transmits non-real-time 1500B messages to *Slave 3*, also generating a load close to 100% of the respective uplink ($M2$). A sniffer is placed in the link of *Slave 3*, allowing capturing and analyzing the downlink traffic.

Figure 6 presents, for each EC, the histogram of the time elapsed between the end of the transmission of the TM and the transmission of both messages, measured during approximately 31 seconds. The horizontal axis represents the timeline of one EC (1ms), with the origin set at the end of the TM transmission. The first $290\mu s$ are reserved for the synchronous window which, in this case, remains empty since no synchronous messages are defined. The asynchronous window, which lasts for $540\mu s$, conveys the asynchronous traffic submitted to the switch, including both by the higher-priority sporadic server, responsible by message stream $M1$, and a lower-priority background server, which uses the remaining bandwidth of the asynchronous window to convey the NRT traffic, which in the present case is restricted to message $M2$.

The histogram clearly shows the higher priority of the sporadic server traffic, which appears at the beginning of the asynchronous window followed by the NRT traffic, using the remaining time. The figure also shows that the server effectively limits the load submitted to the network. In fact, despite the source node submitting a load near 100%, only an average of one packet per EC is effectively forwarded by the switch. Finally, the figure also shows the message confinement, since messages $M1$ and $M2$ are continuously sent, but are only forwarded during the asynchronous window. To better understand the Figure 6, note that the asynchronous window is able to convey only five messages of 1500B. In the depicted experiment the sporadic server is able to send two messages in a row every two ECs. Then the background server fills in the remaining of the asynchronous window with three messages. In the following EC the sporadic server has no budget and thus the background server makes use of the full asynchronous window bandwidth, sending five messages in a row, and then the cycle repeats itself again.

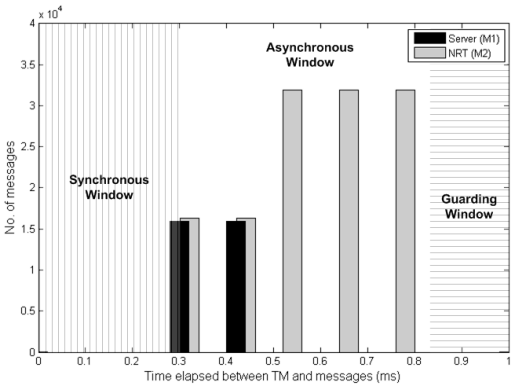

Figure 6. Histogram of transmission inside the EC

The second experiment involves the same nodes but addresses a dynamic scenario in which the real-time traffic load varies at runtime. In this case *Slave 1* sends 150B size real-time messages to *Slave 3* with a variable inter-arrival time that allows controlling the generated load ($M3$). This

stream is handled by the same sporadic server as before. *Slave 2* transmits non-real-time 600B size messages, also with a variable inter-arrival time ($M4$).

Figure 7 presents the throughput curves of each stream in the downlink of *Slave 3*. Initially there is no traffic sent to the switch. At time t=2s *Slave 1* starts the transmission of message $M3$, with an inter-arrival time that starts at 1ms, decreasing over time until it reaches 0, thus generating a load that varies gradually from approximately 1Mbit/s to near 100Mbit/s (100% of its uplink). At time t=12s the transmission of message stream $M4$ is initiated, following a similar pattern as message $M3$, resulting in a load that varies from approximately 5Mbit/s and grows up to near 100Mbit/s. At time t=21s the sporadic server inside the switch reaches saturation at close to 12% load, arising from the 3000B that it can transmit every two ECs. After that point, the extra packets accumulate in the respective FIFO and are eventually discarded. At time t=35s the background server also saturates at close to 41% load for the NRT traffic. This load is the remainder of the asynchronous window after having accounted for the sporadic server traffic. Message $M3$ is suspended at time t=48s, leaving all the asynchronous window available to the NRT traffic. At timet=55s message stream $M3$ is progressively reactivated, as before, causing a decrease in the NRT traffic throughput. The stepwise shape of the curves is due to discrete effects of non-preemptive packet transmission and the arrangement of the packets inside the asynchronous window. The saturation levels are reached again around timet=70s.

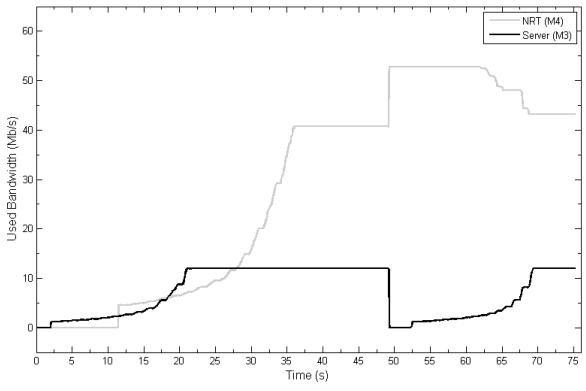

Figure 7. Server and NRT traffic throughputs

This experiment shows the correctness of the hierarchical relation between the top-level polling server, which manages the asynchronous window, and the sporadic and background servers that handle messages $M3$ and $M4$, respectively. Namely it is shown that the background server is able to dynamically reclaim the asynchronous bandwidth not used by the sporadic server, and that both servers altogether never exceed the bandwidth inherited from the predecessor server.

6 Conclusions

Composability is recognized as an appealing property that may bring important reductions in development time and costs of complex embedded systems. Server-oriented architectures are an effective means to support composability by providing resource sharing, in a controlled way, among different application components.

Most complex embedded systems are distributed and thus, composability must also be supported by the networks therein used. A communication technology that is particularly popular in such systems is Ethernet, particularly in different real-time variants. However, the support for server-based traffic scheduling by such protocols is limited in the kind of server policies and hierarchical composition that can be used and in the run-time creation, removal and adaptation of servers.

Recently, the authors proposed the Server-SE protocol to support server-based communication over Ethernet without the constraints referred above. Such protocol, however, operates over COTS switches, thus inheriting a few fundamental limitations these have concerning the lack of control of the timing behavior of message streams.

This paper proposed an implementation of Server-SE over a new customized Ethernet switch that follows the FTT paradigm and which does not suffer from the referred limitations. Particularly, this new Server-SE version over the FTT-enabled switch supports a seamless integration of real-time and non-real-time services, copes with arbitrary traffic arrival patterns, allows arbitrary servers as well as their composition, and supports their dynamic creation and adaption. The paper also describes a prototype implementation as well as a couple of practical experiments that show the protocol capabilities of traffic differentation and confinement.

Acknowledgment

This project was partially supported by the Portuguese Government through grant SFRH/BD/32814/2006 and project HaRTES - PTDC/EEA-ACR/73307/2006 and by the European Community through the ICT NoE ArtistDesign - 214373. The authors also would like to thank Xilinx Inc. for the donation of the Tri-mode Ethernet MAC soft IP core, as well as ISE and ChipScope Pro FPGA design tools.

References

[1] I. Shin and I. Lee, “Compositional real-time scheduling framework with periodic model,” *ACM Trans. Em-*

bed. Comput. Syst., vol. 7, no. 3, pp. 1–39, 2008.

[2] R. Marau, N. Figueiredo, R. Santos, P. Pedreiras, L. Almeida, and T. Nolte, "Server-based Real-Time Communications on Switched Ethernet," in *CRTS 2008: First International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*. Barcelona - Spain: , 2008.

[3] 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, Jun. 2006.

[4] T. Nolte, "Share-driven scheduling of embedded networks," Ph.D. dissertation, Department of Computer and Science and Electronics, Mälardalen University, Sweden, May 2006.

[5] R. Santos, R. Marau, A. Oliveira, P. Pedreiras, and L. Almeida, "Designing a Costumized Ethernet Switch for Safe Hard Real-Time Communication," in *2008 IEEE International Workshop on Factory Communication Systems*. IEEE Computer Society, May 2008, pp. 169 – 177.

[6] 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*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 13–22.

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

[8] TTTech, "TTEthernet," http://www.tttech.com/solutions/ttethernet/, Nov. 2008.

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

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

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