Architectural Solutions for Server Scheduling 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

Abstract

The information exchanged in Network Embedded Systems (NES) is steadily increasing both in terms of quantity, size and complexity. For instance, applications comprising data originated in simple 10 bit ADCs side-by-side with multi-kilobyte variable bit-rate multimedia traffic are, nowadays, becoming a commonplace. Moreover, many NES are frequently subject to real-time constraints and thus the associated information exchanges are subject to timeliness requirements. However, existing real-time Ethernet protocols have difficulties in handling these streams efficiently, particularly in what regards the arbitrary arrival patterns and different QoS requirements.

To overcome these limitations, the authors proposed recently the integration of server-based traffic scheduling concepts within a customizable Ethernet switch, called FTT-enabled switch. The server scheduling unit can be placed in different points of the FTT-enabled switch architecture. The particular placement chosen has a noticeable impact in terms of server responsiveness, flexibility, hardware complexity and global system schedulability.

This paper presents a qualitative comparison about the different architectural solutions and presents a prototype implementation of the hardware-based architecture. Extensive experimental results are also included, showing the correctness of the server operation both in terms of bandwidth guarantees, traffic isolation and latency bounds.

1. Introduction

Switched Ethernet architectures present attractive features such as large bandwidth, cheap network controllers, high availability, easy integration with Internet and a clear path of evolution. These features are fostering the expansion of switched Ethernet architectures to new application areas such as high-speed servoing, target tracking in military systems or even the control of electrical protection systems in substations. However, COTS Ethernet switches are not designed to support the timeliness and safety requirements found in many of the application areas aforementioned due to aspects like blocking caused by long non-preemptive frames, lack of protection against errors in time domain, a limited number of priorities and possible memory overflows.

To address these limitations, diverse Real-Time Ethernet (RTE) protocols have been developed (e.g. [1], [2], [3], [4] [5] [6], [7]). However, most of the RTE protocols share a common difficulty in efficiently handling together real-time messages with diverse arrival patterns, such as periodic and aperiodic, treating them in different ways, frequently with static resource allocation for each case.

Server-oriented architectures are recognized as an effective means to enable such kind of communication resource sharing [8].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. Additionally, 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.

To overcome the limitations mentioned above, the authors proposed previously the Server-SE protocol [9], integrating the FTT-SE [2] and Server-CAN [10] 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,

or environment, without compromising the timeliness of the real-time services. The FTT-SE framework was complemented recently with a costumized Ethernet switch [11] 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.

* Work partially supported by the FCT project PTDC/EEA-ACR/73307/2006 (HaRTES - Hard Real-Time Ethernet Switching)

The architecture of the FTT-enabled switch permits placing the server scheduling unit in different places. More specifically, the server scheduling can be carried out either in software, under control of the FTT Master module, or in hardware, operating complementary to the FTT Master module. This design option has important consequences in terms of responsiveness, flexibility, hardware complexity and global system schedulability. This paper presents the both architectural solutions, performs a qualitative comparison of them regarding the merit figures before mentioned and presents a prototype implementation of the hardware-based approach. Extensive experimental results are also included, showing the correctness of the server operation both in terms of latency bounds, bandwidth guarantees, traffic isolation and hierarchical server composition.

The remaining of the paper is organized as follows: Section 2 presents a brief overview on the related work; Section 3 presents a brief overview about server-based traffic scheduling; Section 4 describes the implementation of software and the hardware-based architecture and discussing their advantages and disadvantages; Section 5 presents experimental results on the hardware implementation and, finally, Section 6 presents the conclusions.

2. Related Work

The nomenclature given to servers in the networking domain is frequently different from the one 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* [1], 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.

Focusing 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 eventually followed by an optional 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. The TTEthernet [7] switch is also based in 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 permits 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 [4]) also presents a TDMA scheme, based on a cyclic bi-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. The protocol is based on a Master-Slave access control scheme, thus servers are scheduled in software. Other protocols, such as [1], 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 Server-SE implementation described in this work.

3. FTT-Enabled Switch

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 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.

In short, the FTT-enabled Switch provides the following advantages: 1) Online admission control, dynamic quality-of-service management and arbitrary traffic scheduling policies; 2) 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; 3) the asynchronous traffic is autonomously

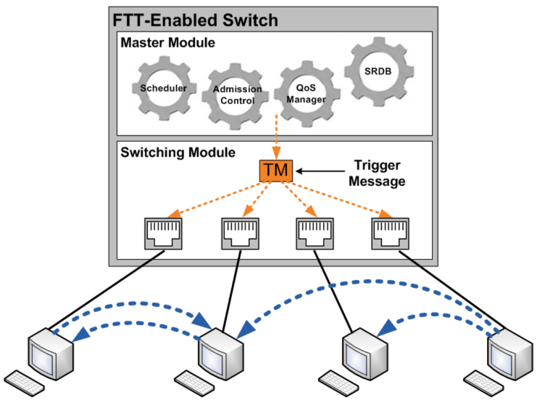


Figure 1. FTT-enabled Ethernet switch.

triggered by the nodes; 4) a seamless integration of standard non-FTT-compliant nodes without jeopardizing the real-time services.

4. Server scheduling integration analysis

The server scheduling in the FTT-enabled switch can be carried either in software, under control of the FTT Master module, or in hardware, operating complementary to the FTT Master module. This design option results in differentiated behaviors in terms of responsiveness, flexibility, hardware complexity and global system schedulability. This section explores both these architectural design options, showing its operation principles and presenting a qualitative comparison among them.

4.1. Servers implemented in software, inside the Master Module

Following a pure software-based approach, the server scheduling can be carried out at the Master node. From the logical operation point of view, this approach is essentially equivalent to the Server-SE protocol [9]. The servers are software entities that reside in the master node. Each server has an associated memory block, organized as a FIFO. The traffic arrives via input ports and is submitted to the Classifier and Verifier Unit that classifies and validates the received messages. Whenever a valid message associated with a given server arrives, it is moved to the respective FIFO. Once every cycle the switch posts the Master about the status of the server FIFOS. Also once every EC the scheduler builds the EC-schedule and generates the trigger message, identifying the messages that should be transmitted in the following EC. The switch intercepts the EC-schedule and then forwards the messages associated with the scheduled servers. This scheme shares essentially the same properties as the Server-SE protocol, providing a great flexibility, permitting the support of arbitrary server schemes as well as its composition, combined with a tight integration with the Master scheduling, admission control and QoS management. However, as seen above, this approach still depends on a explicit signaling mechanism to post the scheduler about the occurrence of server requests.

Additionally, the EC-schedule is built one EC in advance. As illustrated in Figure 3, this whole process results in a server latency between one and two ECs. Thus the server latency is strongly dependent on the EC duration and can be relatively large. This is the cost to pay for having the servers scheduling carried out by the master scheduler in a integrated fashion inside the Master Module.

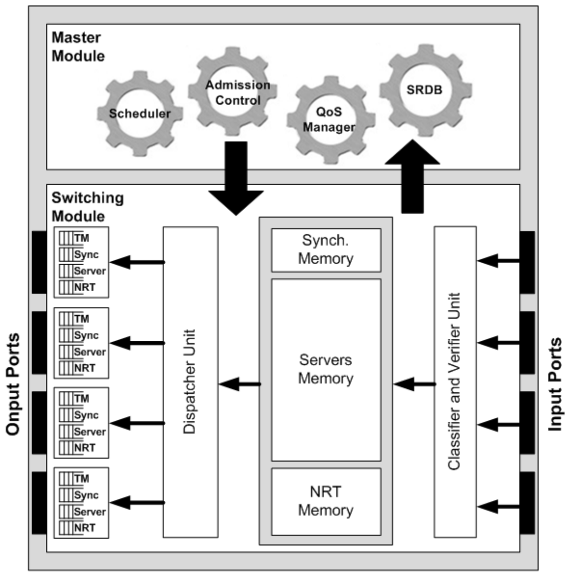
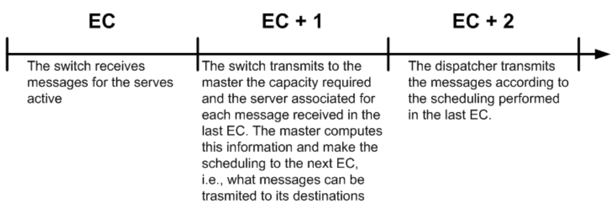


Figure 2. FunctionalArchitecture.



The switch receives messages for the serves active

The switch transmits to the master the capacity required and the server associated for each message received in the last EC. The master computes this information and make the scheduling to the next EC, i.e., what messages can be trasmited to its destinations

The dispatcher transmits the messages according to the scheduling performed in the last EC.

Figure 3. Servers forwarding process.

4.2. Server scheduling implemented inside the Switching Module

The server structures and their scheduling can also be implemented in hardware, inside the Switching Module. The servers are pre-configured and consequently their type and number cannot be changed online. A more dynamic architecture, permitting the dynamic creation and removal of servers, would require online FPGA reconfiguration, a subject that is outside of the scope of this paper. The admission of streams, namely the schedulability analysis and QoS negotiation, continues to be performed inside the Master Module. The negotiation procedure results are then intercepted by the Switching Module and used to configure the servers operational parameters.

Since the server scheduling is carried out independently of the master scheduler, it is necessary to break the EC in two sub-windows, one assigned to the master scheduler and the other to the servers scheduler. Similarly to the software approach, whenever a valid message associated with a given server arrives, it is moved to the respective FIFO. Whenever the server sub-window is reached, the switch checks the server FIFOs, by priority order, and sends any pending messages until either the FIFOs become empty or the server sub-window finishes. This entire process is repeated every EC.

It is straightforward to concluded that, compared with the software architecture, this solution presents a greater reactivity. In the best case the latency is essentially the message transmission time, added with the switching latency, while in the worst case the message arrives at the end of the server sub-window, and thus has to wait to the beginning of the following server sub-window, which takes less than one EC time. Furthermore, for simple scheduling algorithms such as Rate Monotonic or Round-Robin, the implementation is resource-efficient and fairly simple. However, the hardware-based architecture compares negatively to the software one in terms of flexibility. On the one hand the number and type of servers is fixed, as mentioned above. On the other hand, complex servers can require a significant amount of hardware resources. Furthermore, from the global schedulability point of view, the hardware-supported servers also perform worse, since the master and the server scheduler are separate entities, unaware of the state of each other. For instance the master scheduler, when scheduling periodic messages, does not know the state of the servers. Thus, if the number of server requests is not sufficient to fill in the respective sub-window, the master scheduler is unable to reclaim that free space for scheduling periodic messages, thus penalizing the global system schedulability.

5. Experimental Results

The hardware-based server scheduling architecture was deployed in a prototype implementation of a 4 port FTT-enabled Ethernet switch architecture, following a similar Hw/Sw co-design approach as proposed in [?]. The prototype switch implements the Switching Module in hardware, using a NetFPGA board [12], integrating a Virtex-II Pro XC2VP50 FPGA and using 51% of the board total slices, with a maximum operation frequency of 126.20MHz. The Master Module is implemented in software, running in an independent CPU, connected to the FPGA by a dedicated Ethernet link on *Port 4*.

To assess the correct operation of the servers, it was created a configuration with an EC of 1ms, with the servers sub-window using 42% of the EC. Inside the server sub-window are created two sporadic servers, SS1 and SS2, with a budget of 3200B and period 1ms each. In addition it was also created a background server BS, to reclaim the bandwidth left by the sporadic servers. 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 has been analyzed offline, and offers an average load of around 10Mbps, with peaks that may reach 21.9Mbps. The load fed through SS2 is active from the instant t1=22 seconds to t2=58 seconds and, when active, generates a constant load of 48.9Mbps. A simple assessment of the load bandwidth submitted to the servers permits 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 load traffic is served by a lower priority SS2. Since the bandwidth allocated to SS1 and SS2 exceed the server sub-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 the SS2 is active, since the BS has no guaranteed bandwidth.

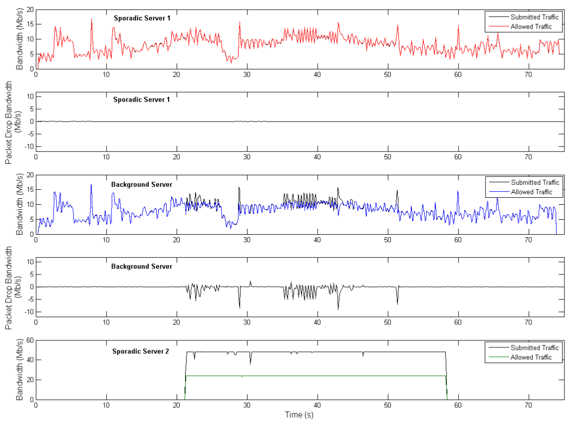


Figure 4. Submitted and forwarded load difference

Figure 4 shows, for each server, the instantaneous input and output traffic bandwidth. The first graph regards the highest priority server SS1 and, in this case, the input and output plots essentially overlap, meaning that the traffic managed by this server is forwarded without a noticeable delay. The second graph respects the video stream served by the BS. Between the instants t1 and t2, corresponding to the instants in which SS2 is active, it is possible to observe several sections in which the input and output traffic plots deviate from each other. The sections in which the input traffic plot is over the output traffic plot corresponds to periods in which the bandwidth allocated to the server is not enough to serve all the input traffic, potentially leading to packet losses. The switch has some buffer capacity, so in some sections the input plot is below the output plot, a situation that corresponds to the points in time where bandwidth allocated to the server is enough to reduce the amount of buffered messages. Finally, the last plot respects the UDP simulated load. As expected, the input and output traffic plots overlap most of the time, with occasional deviations coincident with peaks in the video stream served by SS1.

Figure 5 shows the input and output bandwidth of the three servers between t=32s and t=45s. Here it possible to observe the effect of the relative priority among servers. When SS1 has bandwidth peaks it is possible to observe a corresponding degradation on the BS, while the load traffic

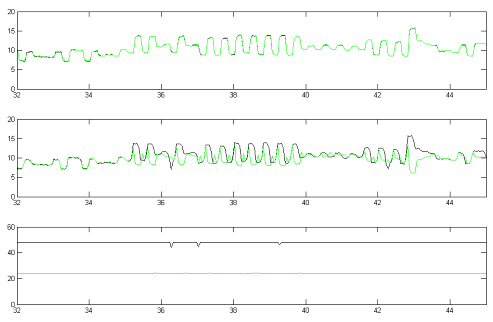


Figure 5. Submitted and forwarded load zoom

served by SS2 is essentially unaffected. This behavior is according with the expectations. SS1 was dimensioned to nearly fit the video stream peak bandwidth, and thus this stream is essentially unaffected by the switch. However, the bandwidth allocated to SS1 and SS2 exceeds the bandwidth of the server sub-window, so when SS1 uses the full bandwidth SS2 capacity is penalized. Finally, the BS receives the bandwidth left over by SS1 and SS2, thus being subject to an higher bandwidth degradation during peak activity of SS1 and SS2.

Table 1 depicts, for each server, the total number of packets transmitted and effectively forwarded by the switch during the experiment. The numeric results confirm the qualitative impressions above enunciated. The highest priority server SS1 experiences a marginal packet loss, while server SS2 experiences a slightly high packet loss ratio. This was expected due to the high bandwidth peaks of the video stream, leading to occasional situations in which the server sub-window capacity is exceeded. Finally, the BS experiences the worst packet loss ration, as expected, since it is the lowest priority server, without any type of guarantees.

	SS1	**SS2**
Packets submitted	stuff	stuff
Packets forwarded	stuff	stuff

Table 1. Number of packets submitted and allowed.

The switch latency was also assessed. For this purpose is was used a configuration similar to the one previously described, except that a single Sporadic Server (SS) received the full server sub-window bandwidth. A packet generator, served by the SS, sent periodically 1370B packets to the switch. The packet generator period was set with a 0.4% offset relative to the EC length, to induce diverse phasing conditions with respect to the server sub-window occurrence. The switch was modified in order to return the packet to the sender, thus enabling the packet generator node to measure the round-trip delay. The obtained results are depicted in Table 2

The best case happens when the message transmission occurs during server sub-window, in which case is forwarded immediately, suffering a round-trip delay due to the packet transmission time added by the processing overhead within the switch. The worst case situation occurs when the transmission is issued right after the end of server sub-window. In this case the round trip delay has to be added up with the time to the beginning of the following server sub-window. With a server sub-window of 42% of the EC and an EC of 1ms, this waiting time is, in worst case, 580ms. Observing Table 2 it is possible to see that the maximum round-trip delay is approximately given by the minimum value added up with 580ms. Thus, the obtained results are consistent, indicating a correct operation of the server scheduling.

	Switch Latency
Minimum	227 μs
Maximum	807 μs
Average	397 μs

Table 2. Number of packets submitted and allowed.

6. Conclusions

Recently, the authors proposed an implementation of Server-SE over a new customized Ethernet switch that follows the FTT paradigm. 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. This paper presents preliminary work on the analysis of the different possibilities of implementation of the server scheduling, and associated tradeoffs, namely in what regards server responsiveness, flexibility, hardware complexity and global system schedulability. This paper also includes a prototype implementation of the hardware-based architecture and its experimental assessment. The experimental results show the feasibility and correctness of the implementation.

References

[1] Loeser, J. and Haertig, H. Low-Latency Hard Real-Time Communication over Switched Ethernet. In *ECRTS '04: Proceedings of the 16th Euromicro Conference on Real-Time Systems*, pages 13–22, Washington, DC, USA, 2004. IEEE Computer Society.

[2] 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, June 2006. IEEE Computer Society.

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

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

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

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

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

[8] Shin, Insik and Lee, Insup. Compositional real-time scheduling framework with periodic model. *ACM Trans. Embed. Comput. Syst.*, 7(3):1–39, 2008.

[9] 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. .

[10] T. Nolte. *Share-Driven Scheduling of Embedded Networks*. PhD thesis, Department of Computer and Science and Electronics, Mälardalen University, Sweden, May 2006.

[11] 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*, pages 169 – 177. IEEE Computer Society, May 2008.

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