Robust and Efficient Server-Based Communication over Switched Ethernet

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

Real-Time Ethernet (RTE) protocols have difficulties in attaining a bandwidth efficient support of aperiodic message streams in scenarios where strict timeliness requirements have to be met. To overcome such difficulties, the authors proposed recently the Server-SE protocol that deploys server-based traffic scheduling over switched Ethernet using the FTT-SE protocol and COTS switches as platform. This paper extends such work by proposing a new platform, namely the FTT-Enabled Switch recently developed by the authors. The resulting framework 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 framework is validated with a prototype implementation. Experimental results show the effectiveness of the framework in guaranteeing a correct temporal behavior even in the presence of traffic with arbitrary arrival patterns and load variations.

1 Introduction

Nowadays, 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 serving, 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. These limitations arise from 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. Some of these protocols, such as traffic shaping [1], master-slave protocols FTT-SE [2], EtherCAT [3], Powerlink [4] and EthernetIP [5] rely strictly on Commercial Of-The-Shelf (COTS) Ethernet components. However, by relying in COTS hardware, the sphere of control of these protocols is limited to the nodes that strictly comply with the associated protocol and the system becomes vulnerable to malfunction nodes. A way to address these problems consists in integrating the traffic management and control mechanisms within the switch itself, creating a modified Ethernet switch. This direction, followed in proposals such as PROFINET [6] and TTEthernet [7], allows obtaining gains in terms of performance and timeliness guarantees. This approach yields a relatively low level of intrusion because it is still possible using COTS hardware and standard software stacks in the end nodes, possibly with specific layers just for accessing the real-time services.

Independently of the particular mechanisms employed, most of the RTE protocols above mentioned 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. In this scope, the authors have previously proposed the Server-SE protocol [8], integrating the FTT-SE [2] and Server-CAN [9] protocols, the former providing a master/slave architecture that supports operational flexibility and the latter providing an integrated server-based traffic scheduling paradigm. This paper extends such previous work, addressing some limitations of the Server-SE protocol, namely the requirement of a cooperative architecture that imposes FTT-type nodes in the network, exclusively, the lack of tolerance to time-domain faults in the end nodes and the need for an explicit aperiodic signaling mechanism that induces some overhead and limits the responsiveness of the protocol.

These aspects are improved by introducing an FTT-Enabled Switch that integratres the FTT master and is capable of confining the incoming traffic to configurable time windows, whichever its type and arrival pattern. The resulting architecture provides a combination of features that are not present on currently available RTE protocols and that

are particularly well suited for supporting open distributed real-time systems. Firstly, there is a seamless integration of real-time and non-real-time services, with strict timeliness guarantees to the first class. Regarding the real-time traffic, the system is extremely flexible, allowing arbitrary (server) scheduling policies and supporting both isochronous as well as sporadic and aperiodic traffic via servers that can be hierarchically composed. Furthermore, the traffic 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. Finally, note that an apreiodic explicit signaling mechanism is no longer needed and the nodes are allowed to transmit such messages autonomously. Thus, any kind of nodes can now be connected to the network without negative impact on the real-time communication.

The paper is organized as follows: Section 2 discusses server-based traffic scheduling in networks; 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* [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.

Regarding specifically to 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 [6], TTEthernet [7] and Ethernet Powerlink [4]), which can be regarded as a combination of several PS. Other protocols, such as [1], implement a traffic shaper that behaves similarly to a DS. However, due to infrastructural limitations, none of these protocols supports arbitrary server policies nor their hierarchical composition. Conversely, the FTT-based Ethernet switch used in this work allows implementing, within flexible and hierarchical partitions, any CPU-oriented server-based scheduling policy, which are vaster and thus provide more options with well studied timing behavior.

3 FTT-enabled Ethernet switch

3.1 Brief overview

The FTT-enabled switch is based on the Flexible Time-Triggered paradigm, with the FTT master included in the switch (Figure 1). The FTT protocol defines three traffic classes: i) periodic real-time messages activated by the master (referred to as *synchronous* since their transmission is synchronized with the periodic traffic scheduler); 2) aperiodic real-time traffic (called *asynchronous*), autonomously activated by the application within each node and 3) non real-time traffic. The synchronous and asynchronous traffic are transmitted within real-time windows and have guaranteed timeliness while the non real-time traffic is scheduled in background, in the time left within the EC. 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.

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

- A performance boost for 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.

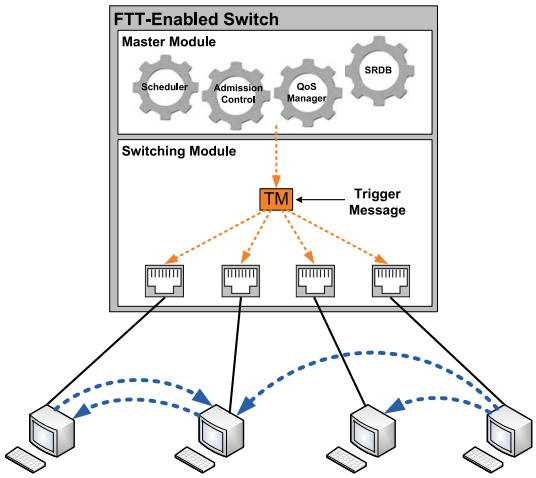

Figure 1. FTT-enabled Ethernet switch.

- 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 [10]. 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 [8] the authors have already addressed the integration of server-based traffic scheduling with the FTT-SE protocol, which operates over COTS switches and which led to the Server-SE protocol. As referred before, this protocol already handles message streams with arbitrary arrival patterns while providing timing guarantees. However, two main limitations subsist that were pointed out before, the requirement for FTT-compliant nodes, only, and the signaling mechanism for aperiodic transmissions. These limitations can be overcome by using an FTT-enabled switch and integrating server-based traffic scheduling mechanisms similarly to what was done with the FTT-SE protocol to put forward Server-SE. Such integration is logically done in the same way, resulting in a similar server hierarchy as depicted in Fig. 2 and presented in [8].

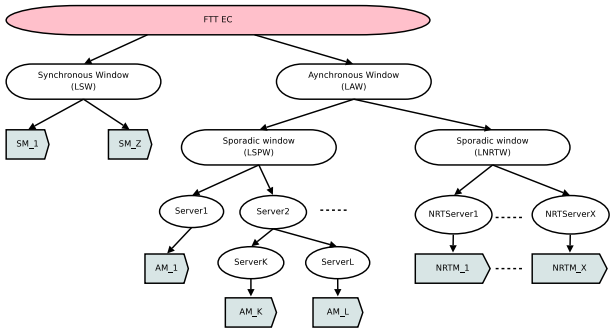

Figure 2. Hierarchy of servers

At the top level the FTT EC structure uses two windows within each EC to handle the two main traffic classes, i.e., synchronous and asynchronous. Both windows are polling servers with period $T_{sw} = T_{aw} = E$ that are currently scheduled in a TDMA fashion. However, they can also be scheduled with fixed priorities with the lower priority asynchronous server reclaiming the space unused by the former. Such scheduling has a higher bandwidth efficiency but also a more complex implementation since the start of the asynchronous window varies from EC to EC. The implementation described further on uses the TDMA approach. The synchronous server has a maximum capacity $C_{sw} = LSW$, resulting in a maximum bandwidth of $U_{max_s w} = \frac{C_{sw}}{T_{sw}} = \frac{LSW}{E}$ for this type of traffic. The latter inherits a maximum capacity of $C_{aw} = LAW = E - LSW - \delta$, with $delta$ being protocol overheads, resulting in a maximum bandwidth of $U_{max_a w} = \frac{C_{aw}}{T_{sw}} = \frac{LAW}{E}$. Note that E and LSW are FTT configuration parameters that can be tuned to suit the global application needs in terms of synchronous and asynchronous requirements. The EC period E establishes the granularity of the remaining servers since their periods are

constrained to be integer multiples of this interval. On the other hand LSW defines the bandwidth shares assigned to each of the two servers at this level and can take any value from 0 to $E-\delta$. Note that 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, which 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 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 is similar to the old implementation. All nodes must negotiate with the switch for the creation of adequate servers in order to handle specific types of traffic. For that, the nodes use a service that relies on specific FTT control messages. On the other hand, the response arrives through the TM, which is sent by the switch, and integrates the appropriate information. The negotiation of the server parameters, i.e, the parameters of the desired virtual channel, is based on admissible ranges of QoS. A suitable negotiation assures 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 resources for further negotiations.

Given the traffic classification and confinement being carried out transparently by the switch, legacy applications can communicate through a virtual channel with QoS guarantees without being aware. In this case, however, an additional entity, e.g., another process in the same or other node, must invoke the creation, negotiation and deletion of the needed servers. Additionally, legacy applications without guaranteed QoS needs can quickly communicate using the NRT background server implemented in each node. This server is created by default and does not require negotiation. In any case, whatever the amount of traffic transmitted by these applications, it does not interfere with the QoS guarantees of the real-time channels already negotiated.

4.2 Proposed functional architecture

Figure 3 presents the functional architecture of the switch with two main modules, the Switching Module and the Master Module. The former is implemented in hardware using an FPGA. The traffic arrives via the input ports 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 is implemented in software and 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.

The integration of these modules can be performed in two ways, the former being currently used:

- The Master Module runs in an independent CPU and the communication with the Switching Module implemented in FPGA is carried out by a conventional communication mean available in the development board (e.g. Ethernet, USB, PCI);
- A processor core embedded in the FPGA, either synthetizable or hardwired (e.g. MicroBlaze, PowerPC), runs the Master Module.

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 Fig. 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 sufficient 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

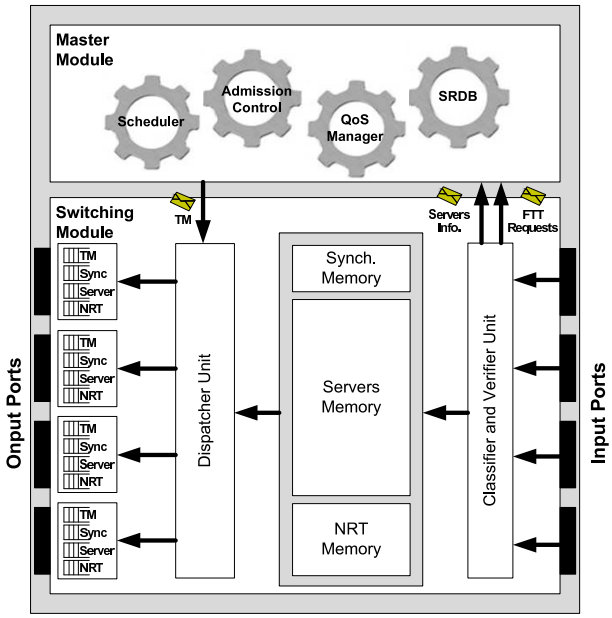

Figure 3. Switch functional architecture

each Elementary Cycle about the messages that were received by the servers, i.e., queued in the respective FIFO, in the previous EC (Servers Info in Fig. 3). With this information the Master knows how much of the servers capacity is requested which is taken into account by the traffic scheduling for the following EC. The result of the scheduling is communicated back to the Switching Module via the Trigger Message (TM) where the Dispatcher Unit enforces the respective transmissions. This process is shown in Figure 4. Its minimum latency is two ECs. This latency is the cost to pay for having the servers scheduling carried out by the Scheduler in an integrated fashion inside the Master Module. Another alternative would be to have the servers being scheduled autonomously in the Switching Module but this would make their dynamic adaptation, e.g., in the scope of dynamic QoS management, more difficult. Nevertheless, this issue is still being analyzed.

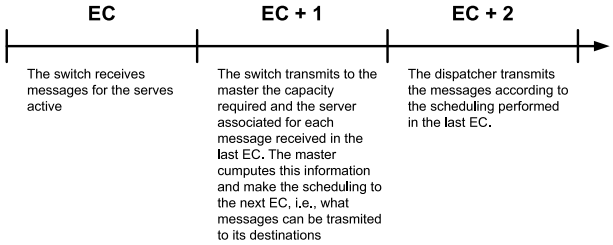

Figure 4. Servers forwarding process

When a communication channel is not needed anymore it is important to close it. 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 some experimental results obtained with a preliminary implementation of the server-based scheduling in the FTT-enabled switch architecture. A NetFPGA board [11] is used that integrates a Virtex-II Pro XC2VP50 FPGA and uses 42% of its total slices with a maximum frequency of operation of 127.13MHz.

The first experiment was carried out integrating different types of traffic in order to validate the switch traffic classification and confinement according to the servers capacities and priorities. Figure 5 illustrates the setup. The Master Module is implemented in a separate computer and communicates with the Switching Module through an Ethernet link connected on *Port 1*. The Master Module implements 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. Making this window as large as the largest asynchronous packet in the network assures that the following TM transmission will never suffer blocking.

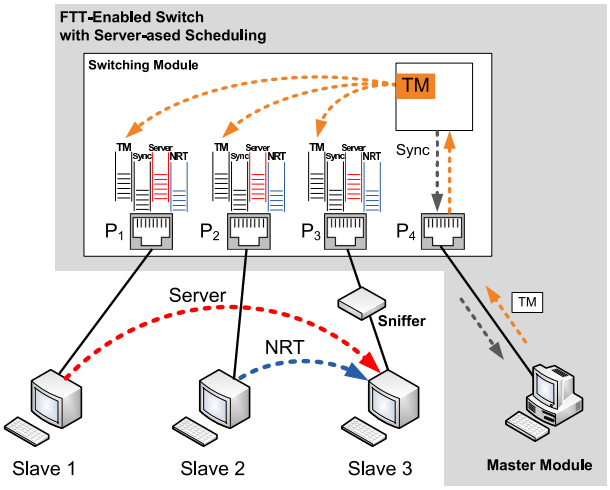

Figure 5. Experimental setup.

In this scenario, we considered one node, *Slave 1*, sending continuously 1500B real-time messages to another node, *Slave 3*, with a minimum inter-frame gap, thus generating a load close to 100% in the respective uplink. A sporadic server was created to handle this stream with 3000B capacity and a period equal to two ECs. Simultaneously, a third node, *Slave 2*, transmits continuously non-real-time

1500B messages to *Slave 3*, also generating a load close to 100% in the respective uplink. A sniffer in the link of *Slave 3* allowed analyzing its downlink traffic.

Figure 6 presents the histogram of the time elapsed between the TM of each EC and the transmission of both messages, the real-time one (Server) and the non-real-time (NRT), measured with the sniffer during approxiamtely 31 seconds. The horizontal axis, thus, represents the timeline of one EC (1ms) with zero at the end of the TM transmission. The first $300\mu s$ are reserved for the synchronous window which, in this case, remains empty. The following window, the asynchronous window, which lasts for approx. $540\mu s$, confines the asynchronous traffic submitted to the switch and it includes the two referred servers, the real-time sporadic server with higher priority, referred to as Server, and the non-real-time one running in background and referred to as NRT.

The histogram clearly shows the higher priority of the sporadic server traffic that appears at the beginning of the asynchronous window followed by the NRT traffic, using the remaining time. Moreover, the sporadic server has a period equal to two ECs and a capacity equal to two packets, meaning that in one EC two packets are sent and none in the following EC. Conversely, the background server uses the remaining time being able to transmit 3 or 5 packets depending on whether the sporadic server has capacity or not. Finally, the EC ends with the guardian window which enforces a blocking-free transmission of the TM. The TM transmission itself occurs in the histogram at the end of this interval.

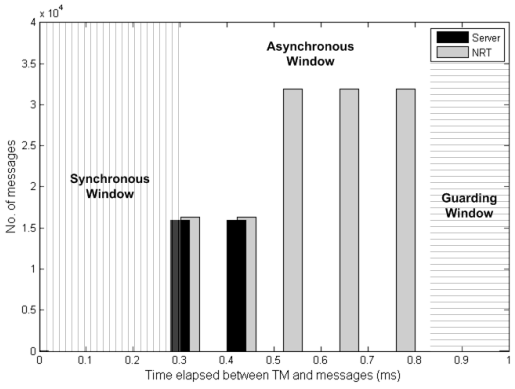

Figure 6. Histogram of transmission inside the EC.

The second experiment involved the same nodes but addressed a more dynamic scenario, with a variable load situation. In this case *Slave 1'* sends 150B real-time messages to *Slave 3* with a variable inter-frame gap that allows controlling the generated load. This stream is again handled by the same sporadic server as before. On the other hand, *Slave 2* transmits non-real-time 600B messages also with a variable inter-frame gap.

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 second 2 *Slave 1* starts the transmission of the real-time traffic with an inter-frame gap of 1ms, decreasing over time until it reaches its minimum possible value and thus generating a load that varies gradually from approx. 1Mbit/s to near 100Mbit/s (100% of its uplink). At second 12 the non-real-time stream is triggered using a variable inter-frame gap in a similar way and starting with a load of approx. 5Mbit/s and growing to near 100Mbit/s. At second 21 the sporadic server inside the switch reaches saturation at close to 12% load, arising from the 3000B that it can transmitt in two ECs. After that point, the extra packets accumulate in the respective FIFO and eventually are discarded. At second 35 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. The server traffic is suspended at second 48, leaving all the asynchronous window in the downlink of *Slave 3* fully available to the NRT traffic. At second 55 the real-time traffic is reactivated progressively as before. This causes a decrease in the NRT traffic throughput. This decrease, however, is not smooth 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 second 70.

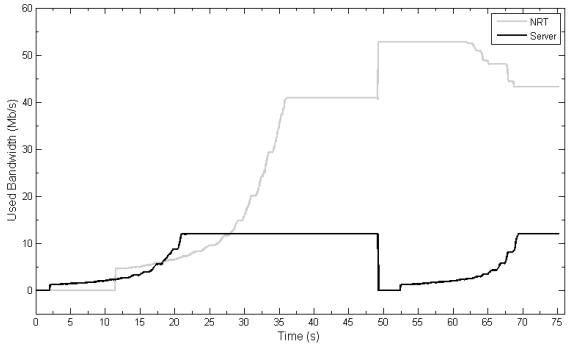

Figure 7. Server and NRT traffic throughputs.

These experiments validate the FTT-enabled switch implementation and particularly the integration of server-based traffic scheduling. They have also shown the capability of the switch to confine different streams using servers, so that even with overloads at the inputs the servers bandwidth is enforced in the outputs. Moreover, the bandwidth reclaiming by the background servers was also shown, even in situations of variable load, which is a rather efficient way

of handling non-real-time traffic without jeopardizing the timeliness guarantees of the real-time one. Finally, the experiments also showed that the traffic control and channels management carried out inside the switch are completely transparent for the nodes that can be heterogeneous and use the network as if they were connected to a COTS switch. We believe this is the first Ethernet switch exhibiting these desireable properties of efficient and robust combination of real-time and non-real-time traffic.

One final aspect that is worth referring is related with the interconnection of several FTT-enabled switches. In fact, since each switch establishes a synchronization domain, their connection requires special care to determine who is addressed by which switch and how the forwarding through multiple switches takes place. This topic, however, is out of the scope of this paper and will be addressed in future work.

6 Conclusions

Server-based traffic scheduling is an advantageous technique to handle heterogeneous traffic types with mutual isolation so that timeliness guarantees can be given even when some streams transmit over their own specification, e.g., in the course of some malfunction. This can be particularly useful to integrate diverse subsystems, legacy applications and support open systems. A preliminary implementation of such technique on switched Ethernet lead to the Server-SE protocol which operated over FTT-SE and COTS switches. However, such implementation suffered from a few limitations arising from the lack of control by COTS switches over the timing behavior of the streams. To eliminate these limitations this paper proposed integrating server-based traffic scheduling inside an FTT-enabled switch. This way, the switch is provided with the needed information to classify and confine all message streams at the inputs, thus enforcing the desired mutual isolation autonomously from the nodes. The paper discussed the integration of the server-based scheduling mechanisms in the FTT-enabled switch and presented experimental results with a prototype implementation that validated the concept. As a result, the proposed switch is highly efficient and robust in handling heterogeneous traffic types with heterogenous nodes, in dynamic environments, being thus adequate to support systems that combine legacy applications and also to support real-time open systems.

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

[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, Jun. 2006.

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

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

[5] O. D. V. Association, "Ethernet/IP," http://www.odva.org/.

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

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

[8] R. Marau, N. Figueiredo, R. Santos, P. Pedreiras, L. Almeida, and T. Nolte, "Server-based Real-Time Communications on Switched Ethernet," in *CRTS 2008: 1st Int Workshop on Compositional Theory and Technology for Real-Time Embedded Systems - satellite of RTSS 2008*, Barcelona - Spain, 2008.

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

[10] 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*, May 2008, pp. 169 – 177.

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