Server-based Real-Time Communications on Switched Ethernet

R. Marau, N. Figueiredo, R. Santos, P. Pedreiras, L. Almeida
DETI / IEETA
Universidade de Aveiro, PORTUGAL
email: {marau,nuno.figueiredo,rsantos,pbrp,lda}@ua.pt

Thomas Nolte
MRTC
Mälardalen University, SWEDEN
email: thomas.nolte@mdh.se

Abstract

Real-Time Ethernet (RTE) protocols have difficulties in the efficient handling of aperiodic message streams with arbitrary arrival patterns, while at the same time supporting the derivation of timeliness guarantees. This paper presents a server-based mechanism for switched Ethernet real-time networks, integrating concepts from the Server-CAN protocol [9] on the FTT-SE protocol [7]. This approach enables an efficient implementation of arbitrary server schedulers as well as their hierarchical composition. Moreover, the presented approach is very suitable for open systems as servers can easily be added, changed and removed during runtime. The paper includes a case study based on a distributed control application. The obtained results illustrate the correct operation of the server-based protocol, showing the capability of the framework in providing strict timeliness guarantees to the real-time traffic in spite of interference with arbitrary arrival patterns and load variations.

1 Introduction

Ethernet has established itself as one of the most important networking technologies, even for systems with tight timing non-functional requirements. Switched Ethernet (SE) architectures, in particular, have been highly regarded for this class of applications since they alleviate the impact of the non-determinism inherent to Ethertons CSMA/CD Medium Access Control (MAC) protocol. Nevertheless the direct use of COTS Ethernet switches still suffers from a few drawbacks that negatively affect their real-time communication capabilities. For example, there can be overflows in port queues with consequences across ports, priority levels and virtual LANs, and the number of priorities are too few for any kind of efficient priority-based scheduling. Moreover, switches present extra latencies and jitter due to the need to interpret frame addresses, and due to different internal architectural solutions [7]. This fact led to an intense research on protocols for providing real-time communication on SE architectures.

Currently available RTE protocols take diverse approaches for providing real-time services. However, despite of the particular mechanisms employed, they share a common difficulty in the efficient handling of real-time messages with different arrival patterns, such as periodic and aperiodic, by treating them in different ways. This paper proposes a different approach, where all types of messages are scheduled in an integrated way, with no distinction between periodic and aperiodic messages. The core of this approach is the integration of the FTT-SE [7] 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 approach facilitates enforcing full control over streams of messages, due to the master/slave operation, no matter of their corresponding arrival patterns, due to the server-based scheduling mechanism. Furthermore, the centralization of the scheduling decisions on the master node also facilitates the implementation of arbitrary server policies as well as their hierarchical composition. This property permits complex applications to be decomposed into sub-applications, each one requiring a share of the bandwidth. Although common in CPU scheduling, this level of flexibility, permitting the implementation of arbitrary server mechanisms as well as their hierarchical composition is, to our best knowledge, new in the context of RTE networks. Summarizing, this new approach for SE systems (1) is free of queues overflows, (2) supports advanced traffic scheduling policies and (3) can enforce real-time guarantees even in the presence of aperiodic communication and/or time-domain faults, e.g., babbling idiots.

The paper starts with a brief overview of related work on real-time Ethernet as well as on server policies for CPU and network scheduling. Then, it proposes using the FTT-SE master/slave protocol [7] to allow a local management of all servers, facilitating their on-line creation, deletion, adaptation and composition. The paper advocates that such a centralized management of the servers provides the required support for open distributed real-time systems as well as for dynamic QoS management. Moreover, this approach also allows any CPU-oriented server-based scheduling policy to
be implemented for network scheduling, possibly with
hierarchical composition, increasing the flexibility of the sys-
tem. The required signaling for the slaves to notify the mas-
ter about their requests is also discussed. Thus, the inte-
gration of the server-based scheduling mechanism within
FTT-SE permits handing of general message streams with
any arrival patterns while confining their impact on the rest
of the system and thus supporting timing guarantees.

The paper is organized as follows: Section 2 presents rel-
ated work; Section 3 revisits the basics of FTT-SE; Section
4 presents the core of the proposal, detailing the integra-
tion of the server-based scheduling mechanism into FTT-
SE; Section 5 presents a prototype implementation, and as-
esses the correct behavior of the framework by means of
a case study based on a control-based testbed; and finally,
Section 6 summarizes the paper and presents some future
work directions.

The paper is organized as follows: Section 2 presents rel-
ated work; Section 3 revisits the basics of FTT-SE; Section
4 presents the core of the proposal, detailing the integra-
tion of the server-based scheduling mechanism into FTT-
SE; Section 5 presents a prototype implementation, and as-
esses the correct behavior of the framework by means of
a case study based on a control-based testbed; and finally,
Section 6 summarizes the paper and presents some future
work directions.

2 Related work

2.1 Real-time communications with SE

This section briefly reviews some of the most relevant
techniques to enable real-time communication on Switched
Ethernet networks. A deeper discussion can be found in
[10].

The first class of approaches includes those enhanc-
ing the switch with traffic control and scheduling capabil-
ities. Hoang et al [5] propose the inclusion of EDF traffic
scheduling and on-line admission control inside a switch.
The EtherCAT protocol [12] presents a similar architecture,
also based on a modified Ethernet switch. PROFINET
Isochronous Real Time (IRT), a new PROFINET real-
time profile [12], employs a distributed cyclic time-slotting
scheme encompassing a deterministic time-triggered phase
and an asynchronous phase for non-real-time traffic. An-
other class of techniques includes those relaying on a traf-
fic shaper in each node to limit the burstiness and amount
of the load submitted to the network and prevent memory
overflows, e.g., as proposed in [6]. Master/slave techniques
have also been proposed. Examples of protocols belong-
ing to this class include the EtherCAT protocol [12], which
uses specialized switches and an open-ring topology and
the ETHERNET Powerlink protocol (EPL) [1], in which
a master node explicitly triggers each transaction according
to a table schedule. Finally, the use of standard switched
Ethernet infrastructures, relying on plain COTS switches,
network interface cards (NIC) and IP stacks, was also con-
sidered, e.g. by Ethernet/IP [2]. Avoiding overloads and
achieving timely behavior in this case requires a careful
analysis by the system designer since there are no run-time
mechanisms to enforce it. The use of traffic shapers [6] is
the approach that relates more closely to the one proposed in
this paper. Shapers are indeed servers that bound the maxi-
mum amount of traffic that a node can send to the network.
However, the typical approach with shapers is to insert them
in a distributed fashion, in all nodes, at the network device
driver level. This distributed approach makes it more diffi-
cult to adjust the properties of the shapers on-line, in an or-
chestrated way, as needed when nodes or streams are added
or removed dynamically. In this paper we propose a central-
ized management of the servers, which efficiently supports
their prompt reconfiguration as needed for dynamic systems
and particularly for dynamic QoS management.

2.2 Server-based CPU scheduling

In the real-time scheduling literature many types of
server-based schedulers have been presented for Fixed Pri-
ority Systems (FPS) and Dynamic Priority Systems (DPS).
These schedulers are characterized partly by the mechanism
for assigning deadlines, and partly by a set of parameters
used to configure the servers, e.g., bandwidth, period and
capacity. The Polling Server (PS) [14] is one of the sim-
plest FPS servers. A PS allocates a share of a resource to
the users of the server. This share is defined by the server
period and capacity. The Deferrable Server (DS) [16] im-
proves the responsiveness of the PS by permitting deferring
its execution whenever there are no user requests. In general
the DS gives better response times than the PS at expenses
of a lower schedulability bound. By changing the way ca-
pacity is replenished for a server, the Sporadic Server (SS)
[14] is a server-based scheduler for FPS systems that allows
high schedulability without compromising too much the re-
sponsiveness.

Examples of Earliest Deadline First (EDF) based DPS
servers include, e.g., the Dynamic Sporadic Server (DSS)
[15]. A very simple (implementation wise) server-based
scheduler that provides faster response-times compared
with SS is the Total Bandwidth Server (TBS) [15]. TBS
makes sure that the server never uses more bandwidth than
allocated to it, yet providing a fast response time to its users
(under the assumption that the users do not consume more
capacity than what they have specified). When the users
desired usage is unknown, the Constant Bandwidth Server
(CBS) [3] can be used, guaranteeing that the server users
will never use more than the server capacity.
2.3 Server-based traffic scheduling

In the network domain, probably for historical reasons, the names given to servers are different. 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, 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 a technique similar to those described in the previous section, based on capacity that is eventually replenished. Many different replenishment policies are also possible, being the periodic replenishment as with PS or DS, the most common. However, it is hard to categorize these network servers similarly to the CPU servers referred in the previous section because networks seldom use clear fixed or dynamic priority traffic management schemes. For example, there is a large variability of Medium Access Control (MAC) protocols, some of them mixing different schemes such as round-robin scheduling with fixed priorities, first-come-first-served, first-come-first-served with multiple priority queues, etc.

In this paper we advocate that, using an adequate protocol, such as one based on the FTT paradigm [4, 7, 11], it is possible to control the traffic in a way that allows implementing any of the CPU-oriented server-based scheduling techniques, which are probably better studied from the timing behavior point-of-view.

3 FTT-SE brief overview

FTT-SE [11] is an RT communication protocol that exploits the FTT master/multi-slave paradigm and the advantages brought by Ethernet micro-segmentation such as parallel forwarding and absence of collisions. The master token, called Trigger Message (TM), is transmitted periodically, creating Elementary Cycles (EC) and polling the slaves to transmit messages. The EC is organized in two consecutive time windows designated synchronous and asynchronous. The former window is associated with the transmission of the synchronous (periodic) traffic, which is triggered by the Master node. The asynchronous window carries asynchronous (aperiodic) traffic, triggered autonomously by the nodes. The asynchronous window is further divided into the real-time (ART) and non real-time (NRT) windows. The ART subwindow corresponds to sporadic traffic, thus has a minimum inter-transmission time \( T_{mit} \) associated to it. This interval is enforced by the Master, i.e., consecutive polls are separated by, at least, that interval. The NRT traffic is handled in background with respect to all the other classes of traffic. Real-time messages (periodic and sporadic) are polled in a per message basis, i.e. the Master explicitly indicates in the TM which are the messages that can be transmitted in the EC. NRT traffic can be handled in a per-message, per-node or per-type basis.

In order to handle aperiodic requests, the FTT-SE protocol uses an efficient signaling mechanism that exploits the full duplex feature of most common switches [8]. Slaves periodically report the current status of their asynchronous queues to the master via status messages sent during the guarding and turnaround windows (Fig. 1). The node’s download links are not part of the TM broadcast path, therefore the signaling messages use time windows that otherwise would be idle, not interfering nor suffering interference from the other messages related with the operation of the protocol, namely the TM and the polled node’s messages.

The signaling mechanism described above allows the master to be aware of the status of all node queues. In each EC the Master processes the status messages, identifies the requests and inserts them in the appropriate scheduling queues. Then, according with the particular scheduling policy implemented, the scheduler decides in which EC such traffic is polled (Fig. 2). The signaling latency \( L_{sig} \) measures the time elapsed between requesting the transmission of a message in a node and the reception of the status message carrying the respective queue status in the Master node. \( L_{sig} \) depends on the number of nodes in the system and on the particular window configuration [8] but, for systems with up to 30 nodes, its value can be between one and two ECs. The polling latency \( L_{pol} \) measures the time elapsed between the transmission of one TM allowing the transmission of one polled message and its effective transmission, and always takes less than one EC. The scheduling latency \( L_{sch} \), not shown in Fig. 2, depends on the scheduling policy used, on the attributes of the specific message being scheduled and on the current load.

4 The Server-SE protocol

The aim of this paper is to develop and present a framework able to handle general message streams in SE with ar-
Handling aperiodic traffic in FTT-SE

Arbitrary arrival patterns while still providing timing guarantees. To achieve this goal it is proposed to integrate server-based scheduling mechanisms with FTT-SE. Hierarchical server composition should also be provided in order to support efficiently the decomposition of complex applications into sub-applications, each with its own share of the bandwidth.

The servers are implemented in the master node, which handles the node requests sent via the FTT-SE asynchronous signaling mechanism and controls the message transmissions by means of the EC-schedule broadcast in the TM. The servers exclusively manage the bandwidth allocated to the asynchronous window. Diverse configurations are possible and, at the extreme, it is possible to have a fully server-managed system by setting the synchronous window length to zero.

4.1 Server allocation and management

At run-time, all nodes must negotiate the creation of adequate servers to handle specific types of traffic with the Master. This negotiation is carried out using asynchronous control channels dynamically created and removed when nodes join and leave the system. The Master answers using the following TM in which it piggybacks the appropriate information, e.g., whether the server was actually created or not. Typically, the communication requirements would be expressed in terms of admissible ranges according to different levels of admissible QoS. These ranges can be used by the master to manage dynamically the QoS of the servers already running, to accommodate new requests.

This mechanism is not transparent for the nodes. For legacy applications it is possible to add a wrapper to carry out the QoS negotiation before initiating the application itself. Alternatively, it is possible to tweak the network driver and to automatically create a background server per node. This server behaves as a “black box”, serving all of the nodes applications without the need to know any details about their respective traffic characteristics. By default this channel is associated with the NRT window. The bandwidth is divided among all nodes and the polling is carried out in a per-node basis. This background server does not provide minimum QoS guarantees, since it uses the bandwidth left available by other higher priority servers, but grants any node some immediate communication capabilities without the need for specific server negotiations. If needed, specific servers can be created latter on for specific traffic.

4.2 Server integration in FTT-SE: architectural overview

The FTT-SE architecture enables a nearly seamless deployment of the server-based management since the master has complete control of the server executions (message transmissions) and knowledge of the status of all queues in all the nodes (via the signaling mechanism). This section describes how the server mechanisms can be integrated in the FTT-SE framework and presents the resulting architecture.

The integration of the server mechanism in the FTT-SE master is carried out by associating one server instance to each asynchronous stream. Each server instance holds the server properties (e.g. budget, period) and manages the server status during runtime. The node’s requests, received via the signaling mechanism above described, are decoded and added to a RequestList. This list is ordered and keeps track of all the pending requests from all the asynchronous messages. The server status dictates whether the requests present in the RequestList are eligible for scheduling or not, thus forming a ReadyQueue from the RequestList. This step makes the server insertion transparent to the Master scheduler, that only has to scan the ReadyQueue and schedule the messages according to the implemented scheduling policy, as if it was the case for normal asynchronous messages.

The integration of the servers in the FTT framework results in the server hierarchy depicted in Fig. 4. At the top level the FTT EC structure divides the traffic into synchronous and asynchronous classes, associated with disjoint windows that fill in the whole EC length. These windows appear once in each EC (period $E$) and have a bounded size ($LSW$ and $LAW$, respectively), specified in the FTT con-
resulting in a (minimum) bandwidth of $U_{\text{sw}}$, it has a (minimum guaranteed) capacity of $C_{\text{sw}} = \text{LSW}$, resulting in a (maximum) workload utilization of $U_{\text{max,sw}} = C_{\text{sw}} = \frac{\text{LSW}}{E}$. On its hand the asynchronous window receives the remaining bandwidth, derived implicitly from the EC size, synchronous window length and protocol overheads [8]. Thus, it has a (minimum guaranteed) capacity of $C_{\text{aw}} = \text{LAW}$, resulting in a (minimum) bandwidth of $U_{\text{max,aw}} = C_{\text{aw}} = \frac{\text{LAW}}{E}$. Note that $E$ and $\text{LSW}$ are FTT configuration parameters that can be tuned to suit the global application needs. The EC period $E$ establishes the granularity of the remaining servers; its periods are constrained to be a multiple integer of $E$. On the other hand $\text{LSW}$ defines, indirectly, the length of the asynchronous window and so how much bandwidth is managed by the servers, or equivalently, the global server budget. Note that $\text{LSW}$ can take any value from 0 to near $E$ ($< E$ due to the protocol overheads), thus the system is highly flexible in this regard.

The second level of the hierarchy manages the sporadic and the NRT traffic. It is implemented as a background server, since sporadic traffic is considered as having real-time requirements and thus is always scheduled before the NRT one. Thus, at this level, the sporadic window inherits the bandwidth and (maximum) capacity of its ancestor server ($C_{\text{spw}} = C_{\text{aw}}; U_{\text{spw}} = U_{\text{aw}}$).

The third level of the hierarchy is where the additional servers can be plugged-in. Arbitrary scheduling policies can be implemented at this level and the sole constraints are that the base time granularity is $E$ and the (minimum) capacity is $C_{\text{spw}}$.

For illustrative purposes the implementation of a Deferable Server is described. This server uses internally a variable, designated activationcounter, that controls the replenishment period. This variable is set to zero upon creation, meaning that the server is eligible for transmitting. Whenever a message is sent, the variable is set to the replenishment period ($\text{init}$) and is decremented every EC until its value reaches zero. This event means that the server is replenished and thus pending, if any, server’s messages present in the RequestList are moved to the ReadyQueue and became eligible for transmission and the process repeats itself. As can be seen, the integration of the server-based mechanism produces minor implications in the FTT-SE architecture. The changes were restricted to the message activation procedure, maintenance of the activationcounter and ReadyQueue management. More complex servers can require additional structures, specially when comprising several messages in a single server scope, similarly to the case of server mechanisms for processor scheduling.

4.3 Limitations and alternative approaches

The Server-SE approach proposed in this paper relies, at the basic level, on the FTT-SE protocol and, consequently, inherits most of the limitations and drawbacks of this protocol. From a conceptual point-of-view the main issues that can be identified in the current approach are the requirement for a cooperative architecture and the explicit signaling mechanism. FTT-SE is a master/multi-slave and thus, as any other master/slave protocol, depends on the cooperation of all the nodes in the system. Malfunction or non-compliant nodes that use the transmission medium without respecting the mediation of the Master can jeopardize all the temporal guarantees. This aspect constraints one of the goals of Server-SE that is to efficiently support open systems, in which nodes can leave and join at will, while still providing timeliness guarantees. On the other hand, the explicit signaling mechanism induces and undesirable latency. Nodes can only report its status in discrete time instants, namely the beginning of the ECs. Thus the signaling latency is, at minimum, 1 to 2 ECs, and can be bigger for systems with a relatively high number of nodes ($> 30$, [8]).

Both of these problems can be solved by an alternative FTT-SE implementation currently under development, which integrates the FTT master within a custom Ethernet switch [13] In this case the FTT-enabled switch confines, by itself, the non-FTT traffic to configurable predefined windows, thus preventing negative effects from non-compliant nodes. Furthermore, the modified switch carries out traffic policing; messages that violate the agreed time domain parameters are trashed and, again, cannot cause negative impact on the correct real-time traffic. Finally, the explicit signaling mechanism is no longer needed and the nodes are allowed to transmit asynchronous messages autonomously. Incoming messages are queued, classified and verified by the switch. Thus, servers can be notified internally on the arrival of new messages, update its status and turn messages eligible for transmission when ap-
propriate. This is on-going work that is currently being explored.

5 Experimental results

This section presents experimental results obtained from a prototype implementation of Server-SE. The first experiment is a basic test meant essentially to verify the correctness of the implementation of the server mechanisms. The second experiment involves a mixed environment composed by a distributed control system, with timeliness requirements, and other sources of traffic, eventually causing overloads. This experiment aims at assessing the suitability of server-based mechanisms to handle time sensitive distributed systems comprising different traffic sources with distinct activation and timeliness requirements.

5.1 Traffic confinement

For this experiment an FTT-SE system was created comprising of one Master node and two slave nodes. The EC duration is set to 1ms and the network operates at 100Mbps. The message set is composed by 3 aperiodic streams with the following properties: \( AM_1 \) and \( AM_2 \) sent by Slave 1 with 2kB and \( mit \) of 6 and 10 ECs, respectively, and \( AM_3 \) sent by Slave 2 with 1kB and \( mit \) 5 ECs. The first two messages were automatically fragmented in 2 packets per instance. Three sporadic servers were set, one for each message. Each server has a budget equal to one instance and a period equal to \( mit \). LAW was constrained to be 240\( \mu \)seconds. In this configuration the servers can schedule no more than 2 packets of the referred messages per EC.

The generation of the message requests was carried out by an application program. Every EC the application produces messages with a probability slightly lower than 1, so that the average rate is slightly below the capacity of the sporadic servers. Figure 5 shows the results for message \( AM_3 \), with the time expressed in ECs. The top histogram shows the inter-arrival times of the requests (signalling messages arrived at the Master). The lower histogram shows the inter-arrival times of message \( AM_3 \) as scheduled by the Master (and transmitted by the respective Slave). The measures were actually carried out within the Master (difference between consecutive requests arrivals and difference between consecutive scheduled transmissions). It is clear that, despite the frequent bursts of consecutive requests, the transmissions respect the \( mit \) corresponding to the respective sporadic server period. The histograms of the other two messages show similar behavior and thus are not presented.

5.2 Case study: distributed control system

This experimental set-up is based on a “ball-on-plane” mechatronic set-up. This control platform aims at keeping one ball as near as possible to a pre-determined point in an horizontal plane. The plane is controlled by two servomechanisms that actuate on two orthogonal axes, X and Y. The ball position and corresponding error is obtained using a video camera connected to a computer.

The platform components form a distributed system based on the Server-SE protocol (Fig. 6). The network bitrate is set to 100Mbps. The platform comprises i) a Sensor Device that captures images and sends them to the controller; ii) a Controller Device that processes the received images, computes the ball coordinates in the plane, executes the control algorithm and sends the setpoints to the actuators (servos) iii) Actuator Devices that receive the commands sent by the controller and position the servos accordingly.

Besides, the sensor, controller and actuator nodes, the system comprises the Server-SE Master node as well as two other computers that simulate a scenario where two surveillance cameras, producing variable load traffic, share the same network. The surveillance cameras also direct the image streams to the controller node in order to overload its network link and cause interference in the control traffic.

The cameras are not synchronized with the network and thus it was decided to use servers to support them. Thus a total of 3 servers, one for each camera, have been created. The controller node was synchronized with the Server-SE network, and thus the control message is configured as a periodic one and sent within the Server-SE synchronous window. In this scope the utilization of servers is important because it guarantees bandwidth isolation (no device can use more bandwidth than the allowed one) and prevents the occurrence of unbounded traffic bursts, guaranteeing a regular and predictable access to the network (upper bounded by the server period). The critical link is, in this case, the one between the switch and the controller. For this reason the
measurements presented in this case study were performed on this link.

In this experiment the control-related traffic was kept constant and the surveillance cameras have been used to simulate a varying load. Eight different experiments, corresponding to increasing link utilization rates, have been carried out. Table 1 summarizes the cameras configuration and global link utilization for each one of the experiments. SC stands for Sensor’s Camera while VC stands for Video surveillance cameras. The utilization value, indicated in the rightmost column of Table 1, includes all the protocol overheads.

<table>
<thead>
<tr>
<th>Test</th>
<th>SC (fps)</th>
<th>VC1 (fps)</th>
<th>VC2 (fps)</th>
<th>Utilization Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30 fps</td>
<td>0 fps</td>
<td>0 fps</td>
<td>19.776%</td>
</tr>
<tr>
<td>2</td>
<td>30 fps</td>
<td>15 fps</td>
<td>15 fps</td>
<td>38.816%</td>
</tr>
<tr>
<td>3</td>
<td>30 fps</td>
<td>29 fps</td>
<td>24 fps</td>
<td>53.414%</td>
</tr>
<tr>
<td>4</td>
<td>30 fps</td>
<td>38 fps</td>
<td>28 fps</td>
<td>61.665%</td>
</tr>
<tr>
<td>5</td>
<td>30 fps</td>
<td>37 fps</td>
<td>38 fps</td>
<td>67.377%</td>
</tr>
<tr>
<td>6</td>
<td>30 fps</td>
<td>42 fps</td>
<td>42 fps</td>
<td>73.089%</td>
</tr>
<tr>
<td>7</td>
<td>30 fps</td>
<td>49 fps</td>
<td>53 fps</td>
<td>84.513%</td>
</tr>
<tr>
<td>8</td>
<td>30 fps</td>
<td>57 fps</td>
<td>57 fps</td>
<td>90.859%</td>
</tr>
</tbody>
</table>

Table 1. Load characterization - Controller downlink.

To assess the impact on the control performance the mean square error of the ball in both the X and Y axes was computed, for each one of the different load conditions. As shown in figures 7 and 8, the control performance remains essentially constant throughout the diverse load conditions. This is a direct result of the presence of the servers, which guarantee the QoS independently of the particular load conditions, preserving the property of composability of the real-time property among streams of messages scheduled through separated servers. Also, the control message, transmitted in the synchronous window, does not suffer from any kind of negative impact due to the surveillance cameras induced overload.

6 Summary and future work

This paper presents a new real-time communication protocol for Switched Ethernet, so called Server-SE, that is capable of handling aperiodic message streams with arbitrary arrival patterns while still providing real-time guarantees. This fact grants a high level of flexibility to this protocol, as well as a high robustness, since it ensures that no overloads will occur within the switch and provides an adequate level of temporal isolation among the streams. Moreover, the protocol supports the dynamic creation, deletion and adaptation of the servers as well as hierarchical server composition. This feature efficiently supports decomposing complex applications in sub-applications, each one requiring a dedicated share of the bandwidth. The impact of the servers is experimentally assessed via a control-based case
study. The experimental results show the effectiveness of the protocol in enforcing guarantees with respect to time-
line as well as providing the agreed service level to real-
time streams, also during overloads. Currently, a number
of scenarios are being evaluated based on different server
policies, and these server-based mechanisms are being in-
tegrated in an FTT-enabled Ethernet switch that will boost
robustness in the sense that the isolation capability will be
integrated in the switch itself, thus not depending on the ex-
istence of specific network device-drivers in the end-nodes.

References

[10] P. Pedreiras and L. Almeida. Approaches to enforce real-
time behaviour in ethernet. In R. Zurawski, editor, The In-