FTT-Ethernet: A Flexible Real-Time Communication Protocol That Supports Dynamic QoS Management on Ethernet-Based Systems

Paulo Pedreiras, *Member, IEEE*, Paolo Gai, Luís Almeida, *Member, IEEE*, and Giorgio C. Buttazzo, *Senior Member, IEEE*

***Abstract*—Ethernet was not originally developed to meet the requirements of real-time industrial automation systems and it was commonly considered unsuited for applications at the field level. Hence, several techniques were developed to make this protocol exhibit real-time behavior, some of them requiring specialized hardware, others providing soft-real-time guarantees only, or others achieving hard real-time guarantees with different levels of bandwidth efficiency. More recently, there has been an effort to support quality-of-service (QoS) negotiation and enforcement but there is not yet an Ethernet-based data link protocol capable of providing dynamic QoS management to further exploit the variable requirements of dynamic applications. This paper presents the FTT-Ethernet protocol, which efficiently supports hard-real-time operation in a flexible way, seamlessly over shared or switched Ethernet. The FTT-Ethernet protocol employs an efficient master/multislave transmission control technique and combines online scheduling with online admission control, to guarantee continued real-time operation under dynamic communication requirements, together with data structures and mechanisms that are tailored to support dynamic QoS management. The paper includes a sample application, aiming at the management of video streams, which highlights the protocol's ability to support dynamic QoS management with real-time guarantees.**

***Index Terms*—Distributed systems, dynamic QoS management, manufacturing automation, real-time communication.**

I. Introduction

NOWADAYS, intelligent nodes, i.e., microprocessor-based communication-enabled devices, are extensively used in the lower layers of both process control and manufacturing industries [44]. In these environments, applications range from embedded command and control systems to image processing, monitoring, human-machine interfacing, etc. The communication among these nodes has specific requirements [4] that are quite different from, and sometimes opposed to, those found in office environments. For instance, predictability is favored against average throughput, and message transmission is typically characterized by time and precedence constraints. Furthermore, the nonrespect of such constraints can have a significant negative impact on the system performance, e.g., degrading the quality of the control action in distributed computer control systems (DCCS) or the quality of the system state observation in distributed monitoring systems (DMS). During the last two decades special-purpose networks have been developed to deliver adequate quality-of-service (QoS) to these systems. They are generically called fieldbuses and are particularly suited for supporting frequent exchanges of small amounts of data under time, precedence and dependability constraints [44]. Some well known examples today include PROFIBUS, WorldFIP, P-Net, Foundation Fieldbus, ControlNet, TTP/C, Controller Area Network (CAN) and CAN-based protocols such as DeviceNet.

In early DCCSs, network nodes presented simple interfaces and supported limited sets of actions. However, the quantity, complexity and functionality of these nodes have been steadily increasing. Consequently, the amount of information that must be exchanged over the network has also increased, for both configuration and operational purposes. The increase on the amount of data exchanged between DCCS nodes is reaching the limits achievable using traditional fieldbuses given their limited bandwidth, typically between 1 and 5 Mbps [4]. Thus, other alternatives are needed to support higher bandwidth demands while fulfilling the main requirements of real-time communication, i.e., predictability and timeliness, implying bounded delays and jitter.

Starting in the 1980s, several high bandwidth general-purpose networks have also been proposed for use at the field level. For example, FDDI and ATM have been extensively analyzed, both for hard and soft real-time communication systems [42]. However, due to high complexity, high cost, lack of flexibility, and interconnection capacity, these protocols have not gained general acceptance [42].

On the other hand, Ethernet, despite also being a general-purpose network, exhibits high availability, low cost, clear path for future expandability and physical compatibility with networks used at higher layers in the factory structures, which is a set of appealing attributes. However, the direct use of Ethernet at the field level was, for many years, impaired by its nondeterministic arbitration mechanism. Thus, several techniques were proposed to overcome such limitation and allow Ethernet to support time-constrained communication. Later on, with the advent of switched Ethernet and its intrinsic absence of collisions, new works have appeared (e.g., [42], [48]) addressing the ability of such a topology to support real-time traffic.

Nevertheless, existing approaches to real-time over Ethernet still exhibit drawbacks either concerning timeliness guarantees,

Manuscript received April 20, 2004; revised May 4, 2005.

P. Pedreiras and L. Almeida are with the Department of Electronics and Telecommunications, University of Aveiro, Aveiro, Portugal.

P. Gai is with Evidence Srl, Pisa, Italy.

G. C. Buttazzo is with the Department of Computer Engineering, University of Pavia, Pavia, Italy

Digital Object Identifier 10.1109/TII.2005.852068

operational flexibility or bandwidth efficiency. Moreover, there is, at the datalink level, a general lack of support for dynamic QoS management with timeliness guarantees as required by emerging automation applications, e.g., industrial multimedia streaming.

This paper presents a new protocol, FTT-Ethernet, which aims at reducing such limitations and fulfilling the requirements of dynamic real-time applications. Particularly, it includes online admission control to guarantee continued real-time operation under dynamic communication requirements, together with data structures and mechanisms that are tailored to support dynamic QoS management.

The rest of this paper is structured as follows. Section II briefly describes the Ethernet protocol and refers some of the most relevant techniques for achieving real-time behavior and QoS support. Section III states the design goals of the FTT-Ethernet protocol and presents its architecture and operation. Section IV describes the protocol implementation using the SHaRK real-time kernel. Section V describes a sample application for assessing and verifying the implementation. Finally, Section VI presents the conclusion.

II. The Way Towards Real-Time Ethernet

Ethernet was invented nearly 30 years ago by Robert Metcalfe at the Xerox's Palo Alto Research Center. Its initial purpose was to connect a personal computer to a laser printer developed by Xerox. Since then, this protocol has evolved in many ways, both concerning transmission speed and topology. In the former aspect, it has grown from the original 2.94 Mbps [5] to 10 Mbps [13], [14], [16], then to 100 Mbps [17] and, more recently, to 1 Gbps [18] and 10 Gbps [19]. Concerning the latter aspect, it moved from a bus topology, initially based on thick coaxial cable [13], to a more structured and fault-tolerant approach based on a star or tree [15], [16]. Despite this evolution, Ethernet protocols have kept two fundamental properties: the existence of a single collision domain (broadcast medium) and the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) arbitration.

According to the CSMA/CD mechanism, the network interface cards (NICs) carry out frame transmission as soon as the application requests it and the bus is sensed free. After starting a transmission, NICs continue sensing the bus for a time slot to detect a collision. If it occurs, all the stations abort the ongoing transmission, issue a jamming signal, and wait for a random time interval before repeating the process. An exponentially increasing wait time is used to reduce the probability of chained collisions on heavy loaded networks.

This nondeterministic nature of the CSMA/CD arbitration mechanism has been considered, for many years, the main obstacle for using Ethernet to support real-time applications. That particular obstacle was removed in the 1990s with the introduction of microsegmented switched Ethernet that bypasses the native CSMA/CD arbitration mechanism and prevents collisions. However, this does not make Ethernet fully deterministic. For example, if a burst of messages destined to a single port arrives at the switch in a short time interval, they must be serialized and transmitted one after the other. If the arriving rate is greater than the transmission rate for a sufficiently large time interval, buffers will be exhausted and messages will be discarded due to buffer overflow. Therefore, even with switched Ethernet, some kind of higher-level coordination is required to enforce timeliness.

A. Brief Survey on Real-Time Techniques

In the quest for real-time communication over Ethernet, several techniques were developed and used by both industry and academia, some of which are briefly referred to in this section, grouped according to their main features. Deeper descriptions of these techniques are presented in [55].

CSMA/CD based protocols achieve real-time behavior over shared Ethernet relying solely on the original CSMA/CD contention resolution mechanism (e.g., NDDS [34], ORTE [40], RTPS [39]), taking advantage of the fact that the probability of collision between concurrent nodes is closely related to the traffic properties such as the bus utilization factor, message lengths and burstiness [38]. A slightly different approach ([23], [24], [27]) consists in shaping the traffic produced by each station, limiting its amount and burstiness. **Modified CSMA protocols** tamper the Ethernet's native CSMA/CD arbitration scheme in order to improve the temporal behavior of the network, either by reducing the probability of collisions (e.g., Virtual Time CSMA protocol [29], [33]) or by sorting out collisions in a deterministic way (e.g., Windows [29], CSMA/DCR [26], and EquB [41]). **Token passing** based mechanisms, allowing only the node in possession of a (unique) token to access the communication medium, have also been proposed (e.g., Timed-Token Protocol [28], RETHER [47], RT-EP [30], [31], and VTPE [3]). **TDMA and Master/Slave** (MS) techniques also enforce mutual exclusion on medium access by, respectively, allocating disjoint time slots to each node (e.g., MARS [22], [25]) and allowing transmissions only after explicit indication of a master node (e.g., ETHERNET Powerlink (EPL) [7], [8]). Token passing, TDMA, and MS techniques completely avoid the possibility of collisions, thus providing deterministic access time to the communication medium.

Since the early 1990s, the interest in **switched Ethernet** has been growing steadily, being a means to improve global throughput, implement traffic isolation, and reduce the impact of the nondeterministic CSMA/CD arbitration. Despite not being capable of providing real-time communication services by themselves [37], switches do reduce the nondeterminism with respect to CSMA/CD medium access control and open the way to efficient implementations of Ethernet-based real-time communication.

As for shared Ethernet, Traffic shaping (e.g., Loeser *et al.* [48]) and MS (e.g., EtherCAT [49]) techniques have been proposed to limit the amount of traffic submitted to the switch, preventing overloads. Another class of approaches consists in *enhancing the switch with extra functionality*, providing more efficient scheduling policies and admission control (e.g., [11], [12], [45], [46] and PROFInet Isochronous Real Time (IRT) [50]). Finally, *standard switched Ethernet infrastructures* may also be used (e.g., [9], [20], [32], and [54]) but avoiding overloads and achieving timely behavior requires a careful analysis by the system designer as there are no run-time mechanisms to enforce them.

B. Dynamic QoS Management

The previous section briefly addressed the most relevant contributions to achieving real-time behavior on Ethernet networks. Such behavior is a requirement imposed by applications typically found at the field or plant level of industrial automation systems. Delivering real-time behavior can be seen as providing an adequate QoS, where quality in this scope stands for guaranteed bandwidth with low and bounded network-induced latency and jitter.

However, many real systems are complex and include different subsystems that operate sporadically, or that endure operational mode changes according to environment stimuli, or that reconfigure dynamically according to online requirements update, or, finally, that need handling a variable number of requests from other subsystems or environments (e.g., flexible manufacturing in which cells reconfigure online according to evolving requirements, mobile robots operating in dynamic environments). In these examples, the level of resources utilized in the system may vary dynamically and, thus, static resource allocation policies become very inefficient. For efficiency reasons, and consequently for cost reasons, such emerging applications require online changes to the communication requirements [43].

In order to support dynamic communication requirements, as referred above, the communication system must feature operational flexibility. On the other hand, the changes in the communication requirements must be carried out without jeopardizing system timeliness. This combination of operational flexibility with timeliness guarantees requires adequate admission control, a feature that is not included in current COTS real-time Ethernet protocols. Conversely, there are a few research protocols that do include admission control, e.g., [11] and [45], and are thus adequate to support online addition of new streams in a timely way. Resource reservation protocol (RSVP) also implements admission control but between Internet Protocol (IP) networks using routers, thus at a layer higher than the one we are addressing in this paper.

However, admission control is still not enough to support dynamic adaptation of message streams, e.g., rate-adaptation [21]. This requires the definition of operational ranges in the message attributes, e.g., minimum and maximum rate, admissible latency, jitter, etc., and use of an online QoS manager that sets the instantaneous individual message parameters maximizing a given reward function [2]. To the best of our knowledge, this level of operational flexibility is not directly supported by any current real-time Ethernet protocol. This has been the major motivation for the proposal of the FTT-Ethernet protocol, which was first introduced in [35] and is presented in Section III.

III. FTT-Ethernet Protocol

Field-level communication systems able to support real-time applications have to fulfill a set of specific requirements. The most commonly referred ones [4] are predictability, support for periodic traffic with different periods, support for sporadic traffic, bounded latency, information on temporal consistency [53], and efficient handling of small data packets.

In addition to those well established requirements, we also consider the need for online admission control and dynamic QoS management to efficiently support the emerging applications referred to in the previous section.

Based on these requirements, we can establish the following goals for a real-time communication protocol.

- Time-triggered communication with operational flexibility.
- Support for on-the-fly changes both on the message set and the scheduling policy used.
- Online admission control and dynamic QoS management.
- Indication of temporal accuracy of real-time messages.
- Support of event-triggered and time-triggered traffic.
- Support of hard, soft, and non real-time traffic.
- Temporal isolation: the distinct types of traffic must not disturb each other.
- Efficient use of network bandwidth.
- Efficient support of multicast messages.
- Use of Ethernet COTS components.

These goals are not fully met by current real-time Ethernet protocols, not only because of the generalized lack of support to dynamic QoS management but also because several protocols either require specialized hardware, or provide probabilistic timeliness guarantees, only, or are bandwidth/response-time inefficient, or, finally, are inflexible concerning the communication requirements, as well as the traffic scheduling policy.

This section presents the FTT-Ethernet protocol that attempts to meet the goals referred above and which is based on the Flexible Time-Triggered (FTT) paradigm [6]. Another implementation of this paradigm is described in [1], namely the FTT-CAN protocol based on CAN.

A. Basic Architectural Options

In order to fulfill the above goals, the protocol uses centralized scheduling and master/multislave transmission control. Since both communication requirements and message scheduling are localized in one single node, the Master, it is possible to update both on-the-fly, thus providing a high level of operational flexibility. This centralization also facilitates implementing online admission control to support dynamic changes in the communication requirements with guaranteed timeliness, as required for dynamic QoS management.

The master/multislave transmission control allows enforcing a coherent notion of time in the network while avoiding collisions since the master explicitly tells each slave when to transmit. It also saves overhead with respect to common master-slave protocols since the same master message is used to trigger several messages in several slaves and the turnaround time in all slaves is overlapped.

FTT-Ethernet, similarly to CAN, WorldFIP [51] and others, uses a source-addressing scheme for real-time traffic. This means that when a message is sent the addressed entity is not the destination but the data item that is being conveyed. This addressing scheme is well suited for control applications, where the data coming from sensors or controllers might be required by several other nodes simultaneously (producer-consumer model [52]). To implement this addressing scheme, the Ethernet frames are transmitted with an adequate destination address, unicast, multicast or broadcast, depending on the

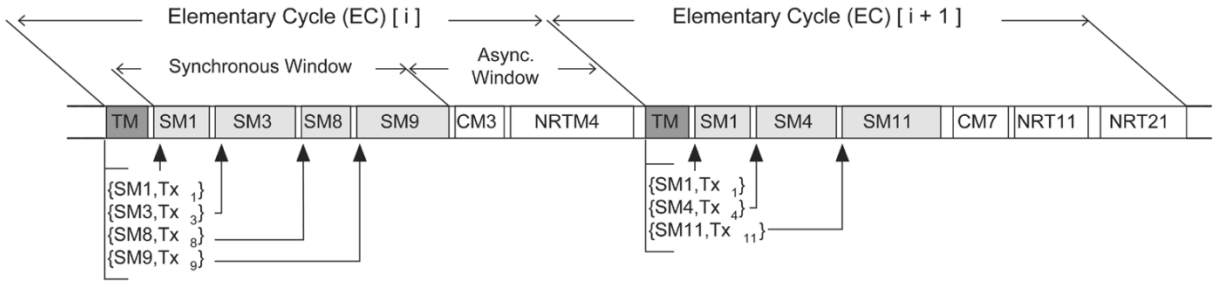


Fig. 1. Elementary cycle structure.

number of consumers for that message being one, a subset, or all, respectively. The message identifier is inserted in the data field. Common Ethernet direct addressing is also used for non-real-time traffic as explained later.

B. Elementary Cycle

A key concept in the protocol is the Elementary Cycle (EC), which is a fixed duration timeslot used to allocate traffic on the bus. This concept is also used in ETHERNET Powerlink [7], and in several other protocols for different media, e.g., WorldFIP. The bus time is then organized as an infinite succession of ECs. Within each EC there can exist several windows reserved to different types of messages. Particularly, two windows are considered: synchronous and asynchronous, dedicated to time-triggered and event-triggered traffic, respectively (Fig. 1).

Each EC begins with the broadcast of a Trigger Message (TM) by the Master node. This control message synchronizes the network and conveys in its data field the identification of the synchronous messages that must be transmitted by the remaining nodes within the respective EC (EC schedule). Moreover, the TM also conveys the information required to allow each node to calculate the time instants within the EC (Tx_i in Fig. 1) at which the synchronous messages should be transmitted. All the remaining nodes on the network decode the TM and scan a local table to identify whether they are the senders of any of the scheduled messages. If so, they transmit those messages in the specified instants.

For the asynchronous traffic, a polling mechanism is used to probe the nodes for the presence of event-triggered messages waiting for transmission. This traffic is divided in two types according to the addressing scheme used: source addressing or direct addressing. The latter is unconstrained traffic, hence non-real-time, that may be associated with common applications using higher-level communication protocols such as TCP/IP (e.g., web server, ftp). The Master node polls the real-time asynchronous traffic first and then, if time is available within the EC, the non-real-time one. The polling order of each of these types of traffic is defined by appropriate scheduling policies, according to the respective communication requirements.

The transmission instants of the messages within the EC are specified in a way that no collisions occur and that no message transmission crosses the boundary of the respective window. Hence, both traffic timeliness within the EC and temporal isolation between all types of traffic are enforced.

C. System Requirements Database

In order to facilitate the management of the communication system, all the relevant operational information is stored locally in the master node, in an appropriate data structure called the System Requirements Database (SRDB). The SRDB contains both the properties of the message streams to be conveyed, as well as all other operational parameters. Specifically, the SRDB contains three components: Synchronous Requirements, Asynchronous Requirements, and System Configuration and Status. The Synchronous Requirements component is formed by the Synchronous Requirements Table (SRT), which includes the description of the synchronous message streams to be conveyed by the communication system:

$$\mathrm{SRT} \equiv \{\mathrm{SM}_i(\mathrm{C}_i\ \mathrm{Ph}_i\ \mathrm{P}_i\ \mathrm{D}_i\ \mathrm{Pr}_i * \mathrm{Xf}_i), \quad i = 1 \ldots \mathrm{N}_s\}$$

where C is the maximum transmission time (including all overheads), Ph the relative phasing (i.e., the initial offset), P the period, D the deadline, and Pr a fixed priority defined by the application. Ph, P and D are expressed as integer multiples of the EC duration. N_S is the number of synchronous messages. $*\mathrm{Xf}$ is a pointer to a custom structure that can be defined to support specific parameters of a given QoS management policy, e.g., admissible period values, elastic coefficient, and relative QoS value (see [21] for a more complete description of using this structure).

The Asynchronous Requirements component is formed by the reunion of two tables, the Asynchronous Requirements Table (ART) and the Non-Real-Time Table (NRT). The ART contains the description of message streams that, despite being transmitted as asynchronous messages, may exhibit time constraints (e.g., alarm messages):

$$\mathrm{ART} \equiv \{\mathrm{AM}_i\ (\mathrm{C}_i\ \mathrm{mit}_i\ \mathrm{D}_i\ \mathrm{Pr}_i), \quad i = 1 \ldots \mathrm{N}_A\}.$$

This table is similar to the SRT except for the use of the mit, minimum interarrival time, instead of the period, and the absence of an initial phase parameter, since there is no phase control between different asynchronous messages.

The NRT contains the information required to guarantee that non-real-time message transmissions fit within the asynchronous window, as required to enforce temporal isolation. The Master only needs to keep track of the length of the longest non-real-time message that is transmitted by each node. The NRT structure is the following:

$$\mathrm{NRT} \equiv \{\mathrm{NM}_i(\mathrm{SID}_i\ MAX\_\mathrm{C}_i\ \mathrm{Pr}_i) \quad i = 1 \ldots \mathrm{N}_N\}$$

where SID is the node identifier, MAX_C is the transmission time of the longest non-real-time message transmitted by that

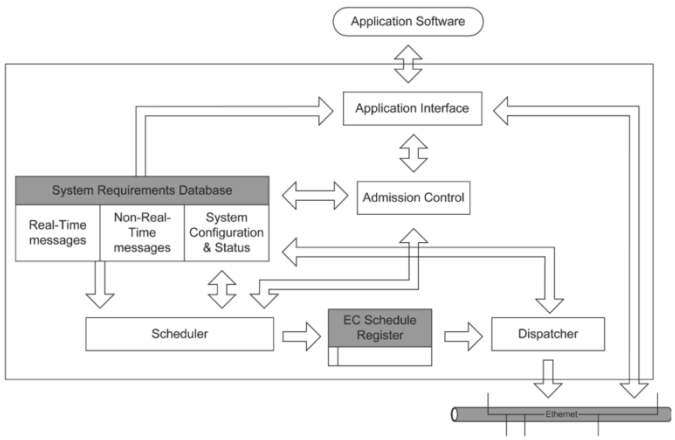


Fig. 2. Master internal architecture.

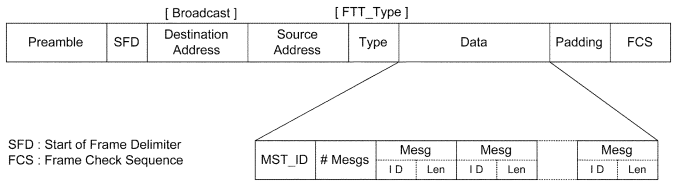


Fig. 3. FTT-Ethernet trigger message.

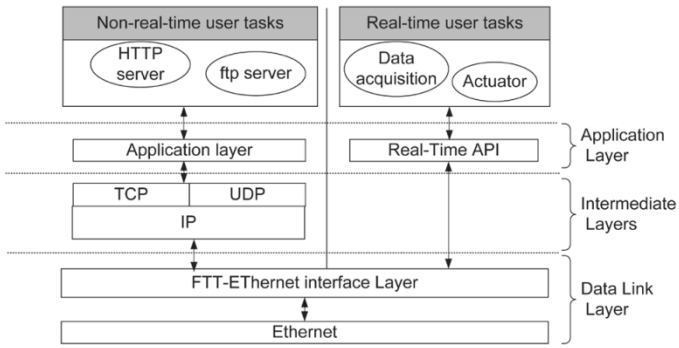


Fig. 4. Internal architecture of slave nodes.

node (including all overheads), and Pr is the node non-real-time priority (used to allow an asymmetrical distribution of the bus bandwidth among nodes). N_N is the number of stations producing non-real-time messages.

The last component of the SRDB is the System Configuration and Status Record (SCSR), which contains all system configuration data plus current traffic figures. This information is made available at the application layer so that it can be used either for profiling purposes or at run-time to make the system adaptive, raise alarms, etc.

D. Master Node Architecture

The Master node plays the role of system coordinator and it is responsible for maintaining the SRDB, building the EC-schedules and broadcasting the EC trigger message.

Fig. 2 depicts the internal architecture of the Master node. The Application Interface provides a set of services to the application software for the SRDB management, regarding the system configuration and to add, modify and delete entries in the SRT, ART, and NRT.

The Scheduler uses the information provided by the SRDB to build the EC-schedules for the synchronous traffic. Each schedule is placed in the EC Schedule Register (ECSR) and consists of the IDs of the synchronous messages to be transmitted in the next EC and their transmission time. The scheduler also gathers information concerning current traffic figures and inserts it in the system status record SCSR.

The Admission Control runs a schedulability test over the synchronous traffic whenever a request for changes in the SRT is made. Changes are admitted upon a positive schedulability result. The Admission Control can be replaced by a QoS Management module to perform QoS negotiation and adaptation over the synchronous message streams, e.g., adjusting online the bandwidth assigned to streams according to the current bandwidth utilization.

The Dispatcher reads the EC Schedule Register, builds the next Trigger Message with such EC schedule, and broadcasts it over the network. Since the EC duration is assumed constant by the scheduler, the TM must be broadcast regularly with sufficient precision.

Both the Scheduler and the Admission Control are encapsulated in modules with clearly defined interfaces. The system supports a seamless integration of several different modules that can be switched online, according to some triggering event. For example, an EDF scheduler can be used and, upon the occurrence of a transient overload, the system can switch to a fixed priority-based scheduling.

Fig. 3 shows the Trigger Message format used in FTT-Ethernet. An Ethernet broadcast Destination Address is used because all the nodes in the network must receive this message. The Type field uses a reserved constant value (FTT Type), used by all messages except the non-real-time ones that are associated with other protocols and use their own Type identifiers. The Data field of the Ethernet frame carries the FTT frame. Its first word identifies the message type (MST_ID for a Trigger Message). The next word contains the number of data messages that should be transmitted in that EC. For each of these messages, the FTT frame contains the respective identifier and transmission duration.

The correct and timely transmission of the EC trigger message is very important for the continued correct behavior of the communication system. Therefore, whenever fault-tolerant communication is required, a mechanism based on redundant masters is used to cope with possible master node failures. Further details on the protocol to enforce replica determinism among redundant masters as well as the bus takeover and handover control are out of the scope of this paper.

E. Slave Nodes Architecture

Slave nodes execute the application software required by the user, eventually requesting the services delivered by the communication system. This system is organized as a stack of layers following the OSI network reference model. However, two parallel stacks are used, one for non-real-time and the other for real-time communication as shown in Fig. 4. The former uses

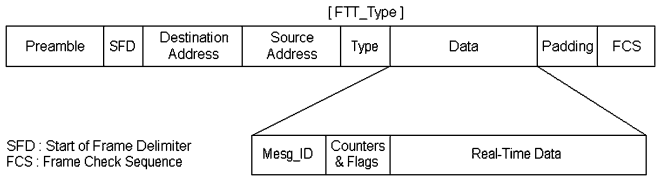


Fig. 5. FTT-Ethernet data message.

a standard IP protocol suite, where the only specific component is a modified Data Link Layer (DLL). The latter follows the collapsed three-layers OSI reference model typically found in fieldbuses. It provides a specific application interface, the Real-Time Application Programming Interface (RTAPI), which allows defining the messages locally produced or consumed, updating and accessing their values and setting up callbacks associated with the respective transmission or reception events.

At the DLL, a transmission control layer is added on top of the Ethernet layer, both for real-time and non-real-time communication. This is referred to as the FTT-Ethernet Interface Layer (Fig. 4), which triggers and manages all communication activities in the system. Particularly, it receives and decodes the Trigger Message, retrieves messages from the network that convey entities requested locally, and, when instructed to do so via the TM, it transmits messages that convey entities produced locally and requested elsewhere. Moreover, it also receives the asynchronous polling requests issued by the master node and transmits the requested traffic.

The non-real-time traffic is intercepted and queued by the FTT-Ethernet Interface Layer. Whenever the node is allowed to transmit this kind of traffic, waiting messages are de-queued and transmitted. On the other hand, all the received non-real-time traffic is passed up to the TCP/IP stack. This method makes the real-time protocol operation fully transparent from the point of view of the non-real-time tasks and messages. Concerning real-time frames, synchronous or asynchronous, their structure is depicted in Fig. 5.

On reception of a data message, the FTT-Ethernet Interface Layer matches the ID of the received message with the list of the locally consumed entities. If it is locally consumed, its local image is updated with the received data. The FTT-Ethernet protocol also supports temporal accuracy information of real-time entities by associating a timer to each such entity. The timer is set with a value corresponding to the validity interval whenever the entity is updated by the application. The timer is decremented while the message waits to be transmitted, and its current value is inserted in the message just before its transmission. On the consumer side, a number of units corresponding to the message transmission time is subtracted from the timer, which then continues being decremented. When the application software consumes a real-time entity, the associated timer value is also delivered together, allowing assessing its temporal accuracy.

F. Performance Issues

FTT-Ethernet belongs to the class of MS protocols, thus inheriting their typical properties (e.g., good timeliness, arbitrary traffic scheduling) and limited bandwidth efficiency. However, FTT is inherently more efficient than other common MS implementations because there is one single control message (TM) per cycle and one common turnaround period, right after the transmission of the TM. With respect to timeliness, adequate schedulability analysis has been developed and presented in [1], [21], and [36]. The protocol uses a fixed duration EC, which reduces the temporal resolution available to express the communication requirements and might lead to a relatively small waste of bandwidth.

FTT-Ethernet also supports asynchronous traffic, handled by polling. This technique is robust with respect to overloads but introduces an extra latency that can be as large as the poll period, plus any related jitter and overhead (Section V-B).

The FTT-Ethernet theoretical efficiency limit is close to the one achieved by TDMA protocols. As examples, the theoretical limit is 67.2% for 1ms ECs at 10 Mbps and using small packets. It raises to 97.5% with 5 ms ECs at 100 Mbps and using large packets. Practical implementations may exhibit lower efficiencies (Section IV), induced by limited accuracy of transmission instants. The current number of possible IDs for synchronous messages is 65 535. FTT-Ethernet does not impose any limit on the number of nodes in one segment neither on the number of messages that can be scheduled in a single cycle (several TMs can be cascaded, if necessary).

G. Vertical Integration Issues in Industry

One aspect that is particularly important in automation systems is vertical integration, meaning that all levels of the industrial organization, from the shop floor to the accountancy, management, and supply chain, must communicate with each other. However, these levels present conflicting communication requirements, a fact commonly referred to as one of the reasons for the proliferation of different industrial communication standards. Ethernet appeared recently as an attempt to support all levels with a single technology, but the different nature of the communication requirements at each level still requires the use of adequate filtering, whichever protocols are used.

In this scope, FTT-Ethernet appears as a protocol for the shop floor and machine control levels, where real-time requirements are more stringent. Within FTT-Ethernet segments, all nodes must comply with the protocol meaning that they must implement the FTT-Ethernet Interface Layer (Section III-E). Thus, to support vertical integration, FTT-Ethernet segments must connect to other segments or levels by means of adequate gateways that respect the protocol transmission control and which can perform the required traffic filtering. These gateways, however, are easy to implement, as no complex protocol conversions are required. An FTT-Ethernet/Ethernet gateway just queues the Ethernet packets and transmits them in the FTT-Ethernet side at the right instants. Moreover, these packets are transmitted without any modification as non-real-time traffic (Section III-E).

H. Microsegmented Switched FTT-Ethernet

FTT-Ethernet can be deployed seamlessly over shared, switched or mixed shared/switched Ethernet networks. However, the use of switches introduces additional latencies that must be taken into account when setting the EC duration and intermessage guarding windows.

When deployed over a microsegmented switched Ethernet, the Master can schedule consecutive synchronous messages without guarding windows in between, since the switch serializes their transmission disregarding any possible overlapping caused by dispatching jitter in the slaves. This boosts bandwidth utilization, as messages are transmitted with minimum interframe gap, and relieves the transmission control system. Also, the traffic scheduling may take into account the destination address of messages, either broadcast, multicast or unicast, exploiting the existence of disjoint paths and consequently increasing the global throughput. Moreover, FTT-Ethernet may enhance the real-time behavior of switches by performing the traffic control required to support adequate management in the output ports queues, for example, preventing overloads and allowing priority scheduling beyond the standard eight priority levels available in IEEE 802.1D. In fact, the scheduling carried out by the Master node may take into account individual priorities of each message, possibly dynamic priorities, e.g., for EDF scheduling, which are neither restricted nor correlated to the eight priority levels defined in IEEE 802.1D.

IV. Prototype Implementation on Shark

Several real-time communication protocols require precise control of the transmission instants and in some other cases a prompt reaction to a command or token. The best way to enforce such capability is to use special-purpose hardware support such as communication controllers that trigger transmissions autonomously. The negative aspect of this choice is that COTS NICs cannot be used any longer, thus raising the costs of the system and reducing the choice of suppliers. Another possibility is to use COTS components and implement the specific protocol layers in software. This option, however, may require adequate software support, normally a real-time operating system (RTOS), to maintain an acceptable control over the transmission instants. This approach is substantially cheaper and easier to implement but typically limits the achievable efficiency because of precision limitations on the timing enforced by the RTOS.

Considering the pros and cons referred to above, it was decided to implement FTT-Ethernet in software. Considering that the protocol includes components that are time-critical and that it is important to reduce to a minimum the potential interference of the application software in the timeliness of the protocol components, it was decided to use a RTOS. These different timeliness requirements are easily managed by the SHaRK real-time kernel [10], through its explicit support of tasks with distinct QoS requirements.

A. SHaRK Brief Overview

SHaRK is a dynamic configurable kernel designed to support hard, soft, and non-real-time applications with interchangeable scheduling algorithms.

The kernel is fully modular in terms of scheduling policies, aperiodic servers, and concurrency control protocols. Modularity is achieved by partitioning the system activities between generic kernel and modules that can be registered at initialization time to configure the kernel according to specific application requirements. The kernel supports device scheduling and thus, it allows extending scheduling algorithms used for the CPU to other hardware resources. Tasks are owned by Scheduling Modules, each of which behaves like a multilevel scheduler in the sense that tasks registered on lower priority modules are scheduled in the background with respect to tasks registered on higher priority modules.

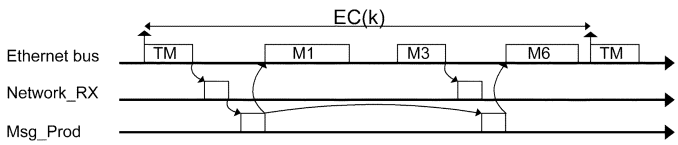


Fig. 6. Time-critical activities (slave nodes).

The system is compliant with almost all the POSIX 1003.13 PSE52 specifications to simplify porting of application code. In addition to the standard features of the previously referred specifications, SHaRK provides various other services, such as

- temporal isolation and task execution time policing;
- asynchronous buffers and other mechanisms for nonblocking intertask communication;
- interrupt and hardware port handling.

B. Implementing FTT-Ethernet on Top of SHaRK

1) Master Node: The internal critical tasks performed inside the master node are two, the Scheduler and the Dispatcher. The Master node also carries other less or non-time-critical activities, such as the system requirements database management, the interface to higher protocol layers, task admission control, the application interface, and the operator interface handling.

The Dispatcher task transmits the Trigger Message, which carries the EC schedule, invoking a SHaRK Network API service that directly sends a packet to the Ethernet layer. The correct behavior of the communication system is linked to the precision in the transmission of the TM; i.e., it must be transmitted with low jitter. Thus, the Dispatcher uses the kernel services for periodic hard tasks and it is assigned to the scheduling module with the highest priority level, preempting every other running tasks.

The Scheduler also has a strict time constraint because it must deliver a new EC schedule before the start of the next EC. For that reason, it is registered as a hard aperiodic task and its execution is triggered by the Dispatcher.

2) Slave Nodes: Correct transmission and reception of Ethernet messages are the critical tasks executed inside the slave nodes. The system also executes other less or nontime-critical activities such as the Local Requirements Database management, the update of the local buffers, the interface to higher protocol layers, and user I/O handling. The critical group includes two tasks, executed in the order depicted in Fig. 6.

Notice that slaves must wait for a TM before initiating any communication activity. Then, every time an Ethernet packet arrives, an interrupt is raised. To control the interference of that interrupt on the currently running task, the network interrupt handler just queues the packet and activates a task, called Network_RX, which afterwards processes the incoming packets. Since the activations of the Network_RX task generally follow an unknown pattern, the respective task model is soft, to bound the amount of processor bandwidth used. Each node becomes

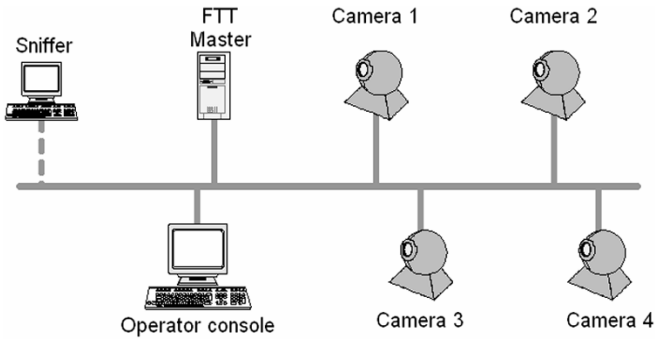


Fig. 7. Experimental setup.

aware of the reception of messages only after the execution of the Network_RX task. Therefore, to limit the end-to-end time from message arrival to the application delivery, this task must be inserted into the highest priority scheduling module.

The reception of a TM activates the Msg_Prod task, which identifies the local synchronous messages that must be transmitted in the current EC and sets a number of timed-events, managed by the kernel. This is the most time-critical and jitter-sensitive task on the slave node since unbounded delays in its execution may lead to delays in the preset transmission instants and, consequently, to collisions. For this reason this task is granted with the highest priority.

V. Application Example: A Remote Surveillance System

To experimentally verify the functionalities of FTT-Ethernet a setup was built using 10 Mbps shared Ethernet. This type of network was chosen because it is more demanding in terms of the system timeliness since collisions must be avoided. It seems, thus, more appropriate for the purpose of verifying the correct operation of the protocol. Similarly, the use of a low transmission rate, 10 Mbps, facilitates the generation of higher bandwidth utilization factors that expose the advantages of using FTT-Ethernet.

The specific application is a video surveillance security system with four independent video cameras, attached to one PC each, the Master station and the Operator console that displays all four images simultaneously (Fig. 7). An additional monitoring station (Sniffer) is used for logging purposes.

The four video streams corresponding to the four cameras are conveyed within synchronous messages. The traffic scheduling policy applied to these messages by the Master station is Earliest Deadline First (EDF).

Each camera can be served with distinct QoS levels, corresponding, in this case, to different frame rates of the respective stream. In normal operating conditions, all the cameras are allowed to send frames at a predefined nominal rate. However, certain events may cause a request for increased QoS of a given camera, e.g., movement detection by an attached sensor or an explicit operator command. If there is not enough bandwidth available to satisfy the request, a management mechanism is applied to try to redistribute the bandwidth among the streams in order to accommodate the request and still deliver acceptable levels of QoS to all streams. If it is not possible to serve a request while delivering at least minimum (prespecified) QoS levels to all streams, the request is rejected, keeping current bandwidth assignments.

TABLE I
Synchronous Message Set Attributes

Camera ID	C_i (μs)	T_{i0} (ms)	$T_{i\ min}$ (ms)	$T_{i\ max}$ (ms)	E_i
1	893	10	5	30	1
2	893	10	5	30	2
3	893	10	5	30	4
4	893	10	5	30	6

This management mechanism is based on the elastic bandwidth management policy [2] according to which, bandwidth requirements are considered to be elastic and can be compressed up to a specified value to cope with an overload condition. Elastic coefficients are used to diversify compression among the streams, so that bandwidth reduction is higher for streams with higher elasticity. The implementation of the elastic bandwidth management mechanism within FTT-Ethernet is detailed in [36].

Using this experimental setup, several measurements were carried out concerning the transmission jitter of synchronous streams and the transmission latency for asynchronous ones. The EC duration was set to 5 ms and the synchronous window was upper bounded to 1.85 ms, i.e., approximately 37% of the EC. This bandwidth was just enough to handle the nominal synchronous requirements. The remaining bandwidth was left for the asynchronous traffic and the transmission of the TM.

A. Synchronous Communication

The synchronous communication requirements arise from the video streams associated with the four cameras. These cameras have a resolution of $384 * 288$ pixels with 8 bits/pixel. The image frames, with approximately 110 KB, are sent in raw format, fragmented in 1 KB packets plus 10 bytes of fragmentation/reassembly overhead. The synchronous message set attributes used in the experiments are shown in Table I, where C_i represents the message transmission time, including FTT-Ethernet overheads, T_{i0}, $T_{i\ \min}$, and $T_{i\ \max}$ are the nominal, minimum, and maximum periods respectively and E_i is the elastic coefficient. The nominal synchronous bandwidth utilization is 35.7%.

At the beginning of the experiments, all cameras send data at their nominal rate. At time $t = 2$ s camera 1 requests an increase in its QoS. The elastic guarantee mechanism finds a feasible solution by increasing the transmission periods of cameras 3 and 4. At time $t = 5$ s, camera 1 returns to its nominal value and the elastic management mechanism resets the message streams to their nominal QoS. Table II summarizes the message periods observed during the experiments.

Fig. 8 presents the number of synchronous packets transmitted by each of the nodes as a function of time. The changes in the rates of the video streams are clearly visible at the instants in which there were requests for QoS changes of camera 1 ($t = 2$ s and $t = 5$ s).

TABLE II
PERIODS OF SYNCHRONOUS STREAMS

Camera ID	Periods of video streams (ms)		
	t≤2s	2<t≤5s	t>5s
1	10	5	10
2	10	10	10
3	10	15	10
4	10	20	10

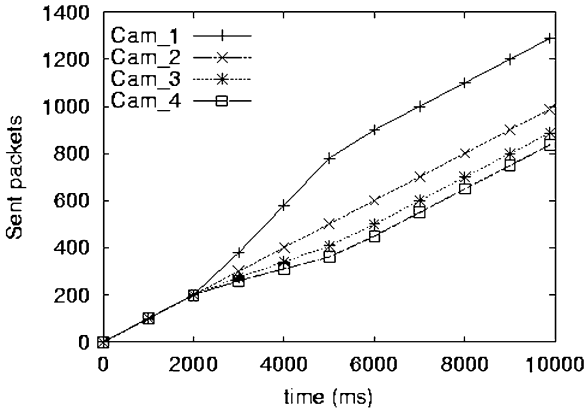


Fig. 8. Packets sent using FTT-Ethernet.

TABLE III
TRANSMISSION JITTER OF SYNCHRONOUS TRAFFIC

Camera ID	1	2	3	4
Avg. relative release jitter (%)	0.53	0.45	1.85	2.83
Absolute release jitter (%)	8.66	7.80	9.79	21.39

TABLE IV
ASYNCHRONOUS MESSAGE SET ATTRIBUTES

Message ID	C_i (μs)	Type	Size (Bytes)	Poll period (ms)
1..4	117	RT	10	5
10	117	RT	30	20
20..23	117	NRT	40	50

Table III summarizes the jitter figures of the synchronous messages sent by the cameras. The values are presented in percentage and normalized to the respective nominal message period. These values are relatively small despite the occurrence of changes in the message set, due to the QoS management carried out by the FTT protocol.

B. Asynchronous Communication

The asynchronous messages used in the experiments include a set of five real- and four non-real-time messages, as depicted in Table IV. Messages 1–4 are real-time alarm messages, used to signal abnormal events, and are polled every 5 ms for fast response. The message with ID 10 is used for QoS management, and consequently its latency restricts the system responsiveness. Therefore, it is classified as real-time and polled every 20 ms. Finally, messages with IDs 20 to 23 are used for service and maintenance. These messages are seldom required and

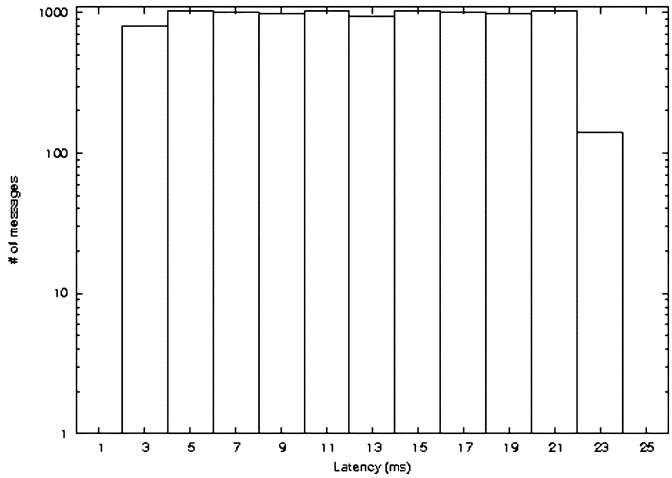


Fig. 9. Asynchronous message latency.

are not subject to time constraints; thus they are configured as non-real-time, with a poll period of 50 ms. Note that, due to its nature, non-real-time messages are transmitted in background with respect to the real-time ones and consequently its polling period is not guaranteed. The resulting asynchronous bandwidth utilization traffic is 10.9%.

The experiments in this case concerned the latency in the transmission of asynchronous real-time messages. For this purpose, the latency affecting the transmission of the QoS message (ID 10) was monitored. This latency was measured between the invocation of the *send()* primitive in the source node and the instant in which the message was inserted in the receiver node queue, being available to the receiver task. The measurements were carried out using timestamps with clocks synchronized to a precision of 10 μs.

Fig. 9 presents a histogram of the latency results obtained with FTT-Ethernet, which vary between 2.2 ms and 22.5 ms, with an average value of 12.3 ms and roughly following a uniform distribution. This is expected, since the ECs are periodic and the asynchronous transmission requests were submitted almost periodically, too, but not synchronized.

The values obtained for the latency are also expected, and can be checked with a trivial response-time analysis. In fact, the worst-case value should correspond to the situation in which the message arrives at the FTT interface layer right after being polled, leading to one polling cycle delay, i.e., 20 ms. Moreover, all higher priority real-time asynchronous messages fit within the asynchronous window and are always polled jointly with this message. Thus, the poll suffers a constant delay but without significant scheduling jitter. On the other hand, the best-case value should occur when the message arrives at the FTT interface layer right before being polled. In that case, the transmission time practically equals the physical transmission of the frame, which is close to 672 μs. Finally, either the best-case or worst-case values must be complemented with the time to buffer the message in the sender node, to wake-up and transmit the message at the right instant, to receive and buffer the message in the receiver node and finally signal the receiving task with new data available. In this experimental setup, these overheads were measured, exhibiting values between 2.0 ms and 2.5 ms.

Therefore, the expected latency values were comprised between 2.6 ms and 23.1 ms, which agree with the measured ones.

VI. Conclusions

This paper addresses the recent trends verified in industrial real-time distributed systems, and particularly at the field level, toward higher flexibility and larger bandwidth requirements. In order to fulfill such requirements, Ethernet has long been proposed for use in such systems. Moreover, several techniques have also been proposed to allow supporting real-time communication. However, these are not yet adequate to support dynamic management of QoS with timeliness guarantees, a feature that is becoming more and more appealing to cope with dynamic environments and run-time updates.

This paper presented the FTT-Ethernet protocol, which has been tailored to specifically support dynamic QoS management under guaranteed timeliness. The basic architectural options have been presented as well as the mechanisms to support dynamic communication requirements, arbitrary traffic scheduling policies, online admission control, time and event-triggered traffic with temporal isolation, and temporal accuracy information.

The paper also presents the protocol implementation using the SHaRK real-time kernel as well as a sample application, for experimental verification of the protocol. The sample application, i.e., a video surveillance system, highlights the capacity of this protocol to support online management of the QoS of several streams.

Finally, as future work, the implementation of FTT-Ethernet over microsegmented switched architectures will be addressed as it may, on one hand, simplify the protocol implementation, and on the other, solve some of the problems that currently affect the real-time performance of Ethernet switches.

References

[1] L. Almeida, P. Pedreiras, and J. A. Fonseca, "The FTT-CAN protocol: Why and how," *IEEE Trans. Ind. Electron.*, vol. 49, no. 6, pp. 1189–1201, Dec. 2002.

[2] G. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni, "Elastic scheduling for flexible workload management," *IEEE Trans. Comput.*, vol. 51, no. 3, pp. 289–302, Mar. 2002.

[3] F. Carreiro, J. A. Fonseca, and P. Pedreiras, "Virtual token-passing Ethernet-VTPE," in *FET'03—5th IFAC Int. Conf. on Fieldbus Systems and their Applications*, Aveiro, Portugal, 2003, pp. 275–282.

[4] J.-D. Decotignie, "A perspective on Ethernet as a fieldbus," in *Proc. FeT'2001—4th Int. Conf. on Fieldbus Systems and Their Applications*, Nancy, France, Nov. 2001, pp. 138–143.

[5] "DIX Ethernet V2.0 Specification,", 1982.

[6] Flexible Time-Triggered (FTT) paradigm. [Online]. Available: http://www.ieeta.pt/lse/ftt

[7] ETHERNET Powerlink Protocol. [Online]. Available: http://ethernet-powerlink.org

[8] (2002) ETHERNET Powerlink Data Transport Services White-Paper Ver. 0005. Bernecker, Rainer Industrie-Elektronic Ges.m.b.H. [Online]. Available: http://www.ethernet-powerlink.org

[9] Ethernet/IP (Industrial Protocol) Specification. [Online]. Available: http://www.odva.org

[10] P. Gai, M. Giorgio, L. Abeni, and G. Buttazzo, "A new kernel approach for modular real-time systems development," in *13th Euromicro Conf. on Real-Time Systems*, Delft, Netherlands, Jun. 2001.

[11] H. Hoang, M. Jonsson, U. Hagstrom, and A. Kallerdahl, "Switched real-time Ethernet with earliest deadline first scheduling—Protocols and traffic handling," in *Proc 10th Int. Workshop on Parallel and Distributed Real-Time Systems*, CITY?, FL, Apr. 2002.

[12] H. Hoang and M. Jonsson, "Switched real-time Ethernet in industrial applications—Asymmetric deadline partitioning scheme," in *Proc. 2nd Int. Workshop on Real-Time LAN's in the Internet Age, RTLIA'03*, Porto, Portugal, Jul. 2003.

[13] *IEEE 802.3 10BASE5 Std.*.

[14] *IEEE 802.3 10BASE2 Std.*.

[15] *IEEE 802.3c 1BASE5 StarLan Std.*.

[16] *IEEE 802.3i 10BASE-T*.

[17] *IEEE 802.3u 100BASE-T*.

[18] *IEEE 802.3z 1000BASE-T*.

[19] *IEEE 802.3ae-2002—10 Gbps*.

[20] J. Jasperneit and P. Neumann, "Switched Ethernet for factory communication," in *Proc of ETFA2001—8th IEEE Int. Conf. on Emerging Technologies and Factory Automation*, Antibes, France, Oct. 2001.

[21] P. Pedreiras and L. Almeida, "The flexible time-triggered (FTT) paradigm: An approach to QoS management in distributed real-time systems. WPDRTS 2003," in *11th IEEE Work. on Parallel and Distributed Real-Time Systems*, Nice, France, Apr. 2003.

[22] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R. Zainlinger, "Distributed fault-tolerant real-time systems: The MARS approach," *IEEE Micro*, vol. 9, no. 1, pp. 25–40, Feb. 1989.

[23] S.-K. Kweon, K. G. Shin, and Q. Zheng, "Statistical real-time communication over Ethernet for manufacturing automation systems," in *Proc. 5th IEEE Real-Time Technology and Applications Symp.*, Jun. 1999.

[24] S.-K. Kweon, K. G. Shin, and G. Workman, "Achieving real-time communication over Ethernet with adaptive traffic smoothing," in *Proc of RTAS'00, 6th IEEE Real-Time Technology and Applications Symp*, Washington, DC, Jun. 2000, pp. 90–100.

[25] J. Lee and H. Shin, "A variable bandwidth allocation scheme for Ethernet-based real-time communication," in *Proc. 2nd Int. Work. on Real-Time Computing Systems and Applications*, Tokyo, Japan, Oct. 1995, pp. 28–33.

[26] G. LeLann and N. Rivierre, "Real-Time Communications Over Broadcast Networks: The CSMA-DCR and the DOD-CSMA-CD Protocols,", INRIA Rep. RR1863, 1993.

[27] L. L. Bello *et al.*, "Fuzzy traffic smoothing: An approach for real-time communication over Ethernet networks," in *Proc of WFCS'02, 4th IEEE Work. on Factory Communication Systems*, Västeras, Sweden, Aug. 2002.

[28] N. Malcolm and W. Zhao, "The timed-token protocol for real-time communications," *IEEE Computer*, vol. 27, no. 1, pp. 35–41, Jan. 1994.

[29] ——, "Hard real-time communications in multiple-access networks," in *Real Time Systems*. Norwell, MA: Kluwer, 1995, vol. 9, pp. 75–107.

[30] J. Martínez, M. Harbour, and J. Gutiérrez, "A multipoint communication protocol based on Ethernet for analyzable distributed applications," in *Proc. 1st Int. Work. on Real-Time LAN's in the Internet Age, RTLIA'02*, Vienna, Austria, 2002.

[31] ——, "RT-EP: Real-time Ethernet protocol for analyzable distributed applications on a minimum real-time posix kernel," in *Proc. 2nd Int. Work. on Real-Time LAN's in the Internet Age, RTLIA'03*, Porto, Portugal, Jul. 2003.

[32] A. Moldovansky, "Utilization of modern switching technology in Ethernet/IP networks," in *Proc. of the 1st Int. Workshop on Real-Time LAN's in the Internet Age, RTLIA'02*, Vienna, Austria, 2002.

[33] M. Molle and L. Kleinrock, "Virtual time CSMA: Why two clocks are better than one," *IEEE Trans. Commun.*, vol. COM-33, pp. 919–933, 1985.

[34] G. Pardo-Castellote, S. Schneider, and M. Hamilton. (1999) NDDS: The Real-Time Publish-Subscribe Middleware. Real-Time Innovations, Inc., Sunnyvale, CA. [Online]. Available: http://www.rti.com/products/ndds/literature.html

[35] P. Pedreiras, P. Gai, and L. Almeida, "The FTT-Ethernet protocol: Merging flexibility, timeliness and efficiency," in *Proc. 14th Euromicro Conf. Real-Time Systems*, Vienna, Austria, 2002, pp. 152–160.

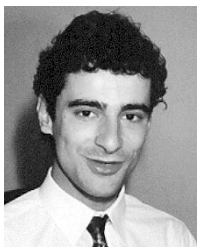
[36] P. Pedreiras, L. Almeida, P. Gai, and G. Buttazzo, "FTT-Ethernet: A platform to implement the elastic task model over message streams," in *Proc. 4th IEEE Int. Workshop on Factory Communication Systems*, Vasteras, Sweden, Aug. 28–30, 2002, pp. 225–232.

[37] P. Pedreiras, R. Leite, and L. Almeida, "Characterizing the real-time behavior of prioritized switched-ethernet," in *RTLIA'03, 2nd Work. on Real-Time LAN's in the Internet Age*, Porto, Portugal, Jul. 2003.

[38] Can Ethernet be Real-Time?. Real-Time Innovations, Inc. [Online]. Available: http://www.rti.com/products/ndds/literature.html

[39] RTPS (Real-Time Publisher/Subscriber Protocol) Part of the IDA (Interface for Distributed Automation) Specification. [Online]. Available: www.ida-group.org

[40] P. Smolik, Z. Sebek, and Z. Hanzalek, "ORTE—Open source implementation of real-time publish-subscribe protocol," in *Proc. 2nd Int. Workshop on Real-Time LAN's in the Internet Age, RTLIA'03*, Porto, Portugal, Jul. 2003.
[41] J. L. Sobrinho and A. S. Krishnakumar, "EQuB-Ethernet quality of service using black bursts," in *Proc. 23rd Conference on Local Computer Networks*, Boston, MA, Oct. 1998, pp. 286–296.
[42] Y. Song, "Time constrained communication over switched Ethernet," in *Proc. FeT'01—4th Int. Conf. Fieldbus Systems and their Applications*, Nancy, France, Nov. 2001, pp. 138–143.
[43] J. Stankovic, C. Lu, H. Son, and G. Tao, "The case for feedback control real-time scheduling," in *Proc. IEEE 11th Euromicro Conf. Real-Time Systems*, 1999, pp. 11–20.
[44] J.-P. Thomesse, "Fieldbus and interoperability," *Contr. Eng. Pract.*, vol. 7, no. 1, pp. 81–94, 1999.
[45] S. Varadarajan and T. Chiueh, "EtheReal: A host-transparent real-time fast Ethernet switch," in *Proc. 6th Int. Conf. on Network Protocols*, Austin, TX, Oct. 1998, pp. 12–21.
[46] S. Varadarajan, "Experiences with ethereal: A fault-tolerant real-time Ethernet switch," in *Proc. 8th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, Antibes, France, Oct. 2001, pp. 184–195.
[47] C. Venkatramani and T. Chiueh, "Supporting real-time traffic on Ethernet," in *Proc of IEEE Real-Time Systems Symposium*, San Juan, PR, Dec. 1994.
[48] J. Loeser and H. Haertig, "Using switched Ethernet for hard real-time communication," in *Proc. Parallel Computing in Electrical Engineering, Int. Conf. (PARELEC'04)*, Dresden, Germany, Sep. 07–10, 2004, pp. 349–353.
[49] EtherCAT Technology Group. [Online]. Available: http://www.ethercat.org/
[50] Real-Time PROFInet IRT. [Online]. Available: http://us.profibus.com/profinet/07/
[51] *European Std. EN 50170. General Purpose Fieldbus: Vol. 1: P-Net; vol. 2: PROFIBUS; vol. 3: WorldFIP. CENELEC, European Committee for Electrotechnical Standardization*, 1996.
[52] T. Li, "Time and industrial local area networks," in *Proc. of COMPEURO'93*, Paris, France, 1993.
[53] H. Kopetz, "Consistency constraints in distributed real-time systems," in *Proc. 8th IFAC Workshop on DCCS*, Vitznau, 1988, pp. 29–34.
[54] M. Schwartz, "Implementation of a TTP/C Cluster Based on Commercial Gigabit Ethernet Components," M.Sc. thesis, TU Wien, Austria, 2002.
[55] P. Pedreiras and L. Almeida, "Approaches to enforce real-time behavior in Ethernet," in *The Industrial Communication Systems Handbook*, R. Zurawski, Ed. Boca Raton, FL: CRC, 2005.



Paulo Pedreiras (M'03) received the licenciatura degree in electronics and telecommunications engineering from the University of Aveiro, Portugal, in 1997 and the Ph.D. degree in electrical engineering, also from University of Aveiro, Portugal, in 2003.

Currently, he is Invited Assistant Professor in the Department of Electronics and Telecommunications, University of Aveiro. Formerly, he was a Design Engineer in a company producing test equipment for the automotive industry. His research interests include distributed embedded systems, real-time networks, real-time operating systems and mobile robotics.

Paolo Gai received the M.S. degree (with honors) in computer science engineering from the University of Pisa, Italy, in 2000, and the Ph.D. in computer engineering from the ReTiS Lab of the Scuola Superiore S. Anna, Pisa, in 2004.

Since 2002 he is CEO of Evidence Srl, a spinoff company of the Scuola Superiore S. Anna. His main research areas are multiprocessor real-time scheduling, memory and code optimization techniques for embedded systems, real-time operating systems for multiprocessor system-on-a-chip especially targeted to automotive applications, real-time networks, and interfaces for the specification of modular scheduling algorithms in real-time operating systems.

Luís Almeida (M'99) received the degree in electronics and telecommunications engineering in 1988 and the Ph.D. degree in electrical engineering in 1999, both from the University of Aveiro, Portugal.

He has been an Assistant Professor with the Department of Electronics and Telecommunications, University of Aveiro, since 1999. He is also a Senior Researcher at the IEETA research unit of the same university. Formerly, he was a Design Engineer in a company producing digital telecommunications equipment.

His research interests lie in the fields of real-time networks for distributed industrial/embedded systems and navigation control for mobile robots.

Giorgio C. Buttazzo (M'93–SM'05) graduated in electronic engineering from the University of Pisa, Italy, in 1985, received the M.S. degree in computer science from the University of Pennsylvania, Philadelphia, in 1987, and the Ph.D. degree in computer engineering from the Scuola Superiore S. Anna of Pisa, Italy, in 1991.

He is an Associate Professor of computer engineering at the University of Pavia, Italy. His main research interests include real-time operating systems, scheduling algorithms, quality of service control, multimedia systems, advanced robotics applications, and neural networks.

Dr. Buttazzo is a Senior Member of the IEEE Computer Society.