FTT-Ethernet: a platform to implement the Elastic Task Model over message streams

Paulo Pedreiras*, Luis Almeida
pedreiras@alunos.det.ua.pt, lda@det.ua.pt
DET / IEETA - Universidade de Aveiro
Aveiro, Portugal

Paolo Gai
pj@sssup.it
Retis Lab - Scuola Superiori Sant'Anna
Pisa, Italy

Giorgio Buttazzo
buttazzo@unipv.it
DIS - Università di Pavia
Pavia, Italy

Abstract

Real-time distributed systems are becoming more pervasive, supporting a broad range of applications such as automotive, adaptive control, robotics, computer vision, and multimedia. Furthermore, in all such applications there is a growing demand for flexibility in order to support dynamic configuration changes such as those arising from evolving requirements and on-line Quality-of-Service management. The elastic task model, proposed previously, is well suited to support that level of flexibility in multitasking systems running on single processors. This paper presents the extension of such model to the network, which runs the FTT-Ethernet protocol. The paper includes a brief presentation of this protocol and of the elastic task model, discusses the referred extension and presents a set of experimental results involving the dynamic adjustment of the quality of service delivered to several message streams, with guaranteed timeliness.

1. Introduction

Real-time distributed systems are becoming more and more pervasive in many different domains of modern societies, from industry to business, health, leisure and others. They are used to support a broad range of applications such as process control, factory automation, automotive, robotics, computer vision, and multimedia. Furthermore, in all such applications there is a growing demand for flexibility in order to support dynamic configuration changes such as those arising from evolving requirements and on-line Quality-of-Service (QoS) management. However, reconciling flexibility and timeliness is not always an easy task. Most existing systems privilege either one or the other but not both [1].

One of the aspects of flexibility that has received recent attention from the research community is the capability of changing on-line the computing and communication requirements of the system under guaranteed timeliness. For example, [6] proposed a load scaling technique to degrade the workload of a system by adjusting the task periods. Tasks are assumed to be equally important and the objective is to minimize the number of fundamental frequencies to improve schedulability under static priority assignments. In [7], Lee, Rajkumar and Mercer proposed a number of policies to dynamically adjust tasks' rates in overload conditions. In [9], Nakajima showed how a multimedia activity can adapt its requirements during transient overloads by scaling down its rate or its computational demand. However, it is not clear how the QoS can be increased when the system is underloaded. In [2], Beccari et al. proposed several policies for handling overload through period adjustment. The authors, however, do not address the problem of increasing the task rates when the processor is not fully utilized. In [3] and [5], Buttazzo et al. propose the elastic task model according to which task utilizations are treated like springs that can adapt to a given workload through period variations. The advantage of the elastic model with respect to the other methods proposed in the literature is that a new period configuration can easily be determined on line as a function of the elastic coefficients, which can be set to reflect tasks' importance. Once elastic coefficients are defined based on some design criterion, periods can be quickly computed on line depending on the current workload and the desired load level. Moreover, the elastic model can also be used in combination with a feedback mechanism, as done in [4], when system parameters are not known a priori.

In this paper, the elastic task model is extended to message scheduling over FTT-Ethernet (Flexible Time-Triggered communication over Ethernet) [10]. This extension allows varying the QoS delivered to different periodic message streams by controlling their transmission periods. The variation in QoS is achieved in a way that improves the efficiency in network bandwidth utilization. Furthermore, the paper also shows experiments with a distributed multimedia application. The respective results confirm the interest in using FTT-Ethernet instead of plain Ethernet or Switched-Ethernet

*This work was partially supported by the Portuguese Government through grant PRAXIS XXI/BD/21679/99, project CIDER POSI/1999/CHS/33139 and by the European Comission through accompanying measure ARTIST IST-2001-34820.

in what concerns message losses and network-induced jitter.

The paper is organized as follows. Section2 introduces the elastic task model. Section 3 presents a brief description of FTT-Ethernet. Section 4 discusses the extension of the elastic task model to message scheduling in FTT-Ethernet. Section 5 presents the experimental results and Section 6 concludes the paper.

2. Elastic task model

According to the classical elastic model proposed in [3], the utilization of a task is treated as an elastic parameter, whose value can be modified by changing the period within a specified range. Each task is characterized by five parameters: a worst-case computation time C_i, a nominal period T_{i_0}, a minimum period $T_{i_{min}}$, a maximum period $T_{i_{max}}$, and an elastic coefficient E_i.

The elastic coefficient specifies the flexibility of the task to vary its utilization for adapting the system to a new feasible rate configuration: the greater E_i, the more elastic the task.

Thus, an elastic task is denoted as:

$$\tau_i(C_i, T_{i_0}, T_{i_{min}}, T_{i_{max}}, E_i).$$

From a design perspective, elastic coefficients can be set equal to values which are inversely proportional to tasks' importance. In the following, T_i will denote the actual period of task τ_i, which is constrained to be in the range $[T_{i_{min}}, T_{i_{max}}]$. Any period variation is always subject to an *elastic* guarantee and is accepted only if there exists a feasible schedule in which all the other periods are within their range. In such a framework, tasks are scheduled by the Earliest Deadline First algorithm [8]. Hence, if $\sum \frac{C_i}{T_{i_{min}}} \leq 1$, all tasks can be created at the minimum period $T_{i_{min}}$, otherwise the elastic algorithm is used to adapt each task τ_i period to some T_i such that $\sum \frac{C_i}{T_i} = U_d \leq 1$, where C_i is the actual on-line execution estimate and U_d is some desired utilization factor.

In the absence of period constraints (i.e., if $T_{max} = \infty$) the period T_i of each compressed task can be computed as follows:

$$\forall i \quad T_i = T_{i_{min}} - (U_0 - U_d)\frac{E_i}{E_v}. \tag{1}$$

where U_0 is the sum of tasks nominal utilizations and

$$E_v = \sum_{i=1}^{n} E_i. \tag{2}$$

In the presence of period constraints ($T_i \leq T_{i_{max}}$), however, the problem of finding the values T_i requires an iterative solution. In fact, if during compression one or more tasks reach their maximum period, the additional compression has to affect only to the remaining tasks. Thus, at each instant, the set Γ of tasks can be divided into two subsets: a set Γ_f of fixed tasks having maximum period, and a set Γ_v of variable tasks whose period can still be enlarged. Applying the equations to the set Γ_v of variable springs, we have

$$\forall \tau_i \in \Gamma_v \quad U_i = U_{i_0} - (U_{v_0} - U_d + U_f)\frac{E_i}{E_v} \tag{3}$$

where

$$U_{v_0} = \sum_{\tau_i \in \Gamma_v} U_{i_0} \tag{4}$$

$$U_f = \sum_{\tau_i \in \Gamma_f} U_{i_{min}} \tag{5}$$

$$E_v = \sum_{\tau_i \in \Gamma_v} E_i. \tag{6}$$

If there exist tasks for which $U_i < U_{i_{min}}$, then the period of those tasks has to be fixed at its maximum value $T_{i_{max}}$ (so that $U_i = U_{i_{min}}$), sets Γ_f and Γ_v must be updated (hence, U_f and E_v recomputed), and equation (3) applied again to the tasks in Γ_v. If there exists a feasible solution, that is, if the desired utilization U_d is greater than or equal to the minimum possible utilization $U_{min} = \sum_{i=1}^{n} \frac{C_i}{T_{i_{max}}}$, the iterative process ends when each value computed by equation (3) is greater than or equal to its corresponding minimum $U_{i_{min}}$. In[5] it is shown that, in the worst case, the compression algorithm converges to a solution (if there exists one) in $O(n^2)$ steps, where n is the number of tasks.

The same algorithm can be used to reduce the periods when the overload is over, so adapting task rates to the current load condition to better exploit the computational resources.

3. FTT-Ethernet protocol brief presentation

The FTT-Ethernet protocol has been firstly presented in [10]. However, it is an implementation on Ethernet of the same paradigm that lead previously to the creation of the FTT-CAN protocol (Flexible Time-Triggered communication on Controller Area Network) [13]. The rationale behind these protocols is the combination of scalability, composability, flexibility, timeliness and efficiency. To achieve these goals the protocols rely on two main features: centralized scheduling and master/multi-slave transmission control. The former feature allows having both the communication requirements as well as the message scheduling and dispatching policy localized in one single node called Master, facilitating on-line changes to both. As a result, a high level of flexibility is achieved. On the other hand, such centralization also facilitates the implementation of on-line admission control in the Master node to guarantee the traffic timeliness upon requests for changes in the communication requirements. In particular, master/multi-slave transmission control, allows to enforce the traffic timeliness in the bus and to achieve a high efficiency in bandwidth utilization. The first aspect is typical of master-slave transmission control since the master explicitly tells each slave when to transmit, thus enforcing the traffic timeliness. The second aspect results from the fact that, instead of using a master-slave transmission control in a per message basis, the same master message is used to trigger several messages in several slaves, thus reducing the number of control messages and consequently improving the bandwidth utilization.

3.1. The Elementary Cycle

A key concept in the protocol is that of the Elementary Cycle (EC). This is a fixed duration (LEC) time-slot, used to allocate traffic on the bus. 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 (Figure 1). The former type of traffic, synchronous, is subject to admission control and thus its timeliness is guaranteed, i.e. real-time traffic. The latter type, asynchronous, is based on a best effort paradigm and aims at supporting event-triggered traffic. If required, it is possible to pre-analyze its requirements and support real-time event-triggered communication.

The traffic in both windows is managed by two complementary subsystems, the SMS (Synchronous Messaging System) and AMS (Asynchronous Messaging System). The bandwidth used by the SMS can be upper bounded by limiting the maximum duration of the respective window to a value (LSW) shorter than the EC duration. However, in each EC, the synchronous window only takes the duration required to convey the synchronous messages that are scheduled for that EC. The remaining time is absorbed by the asynchronous window. Consequently, limiting the maximum bandwidth available for the SMS implicitly causes a minimum bandwidth that is guaranteed to be available for the AMS.

Each EC begins with the transmission, by the Master node, of a Trigger Message (TM). This is a control message that 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 figure 1) at which the synchronous messages should be transmitted. All the remaining nodes in 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.

As far as the asynchronous traffic is concerned, a polling mechanism is used to question 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 former is used for messages that may have time constraints, (e.g., alarm messages, sporadic data, change requests for the synchronous messages). The latter is unconstrained traffic, consequently non-real-time, that may be associated to common applications using higher-level communication protocol,s such as TCP/IP (e.g., web server, ftp). The master node polls the real-time asynchronous traffic first and then, if there is time available within the EC, the non-real-time. The polling order of these types of traffic is defined by appropriate scheduling policies, according to the respective communication requirements.

The transmission instants of all 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.

3.2. The System Requirements Database (SRDB)

In order to facilitate the management of the communication system, all the relevant operational information is stored locally in an appropriate data structure in the master node called System Requirements Database (SRDB). The SRDB contains the properties of the message streams to be conveyed as well as all other operational parameters, e.g., EC duration, maximum width of the synchronous window, maximum number of synchronous message streams, current figures from on-going traffic such as network-induced jitter and deadline misses.

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) that includes the description of the synchronous message streams to be conveyed by the communication system. The structure of this table depends on the particular scheduling policy that is implemented. For the case of the Elastic Task Model, its structure is as follows (7):

$$SRT \equiv \{SM_i(DLC_i,\ C_i,\ T_{i_0},\ T_{i_{min}},\ T_{i_{max}},\ E_i),\ i = 1..N_S\} \quad (7)$$

Each synchronous message stream is characterized by the following parameters: a data length in bytes DLC_i, its respective maximum transmission time (including all overheads) C_i, a nominal period T_{i_0}, a minimum period $T_{i_{min}}$, a maximum period $T_{i_{max}}$ and an elastic coefficient E_i.

The Asynchronous Requirements component if formed by the reunion of two tables, the Asynchronous Requirements Table (ART) and the Non-Real-Time Table (NRT). The former one contains the description of message streams that, despite being transmitted as asynchronous messages, may have time constraints (e.g. alarm messages) (8).

$$ART \equiv \{AM_i(DLC_i,\ C_i,\ mit_i,\ D_i,\ Pr_i),\ i = 1..N_A\} \quad (8)$$

As in (7), DLC_i and C_i are the data length and maximum transmission time of the message, respectively. mit_i specifies the minimum inter-arrival time, D_i the message deadline and Pr_i a static priority.

The NRT (equation 9) contains the information required to guarantee that the transmission of these non-real-time messages fits within the asynchronous window, as required to enforce temporal isolation. Thus, the Master only needs to keep track of the length of the longest non-real-time message that is transmitted by each node. A priority field is still used in order to allow an asymmetrical distribution of the bus bandwidth among nodes, if desired.

$$NRT \equiv \{NM_i(SID_i,\ MaxDLC_i,\ MaxC_i,\ Pr_i),\ i = 1..N_N\} \quad (9)$$

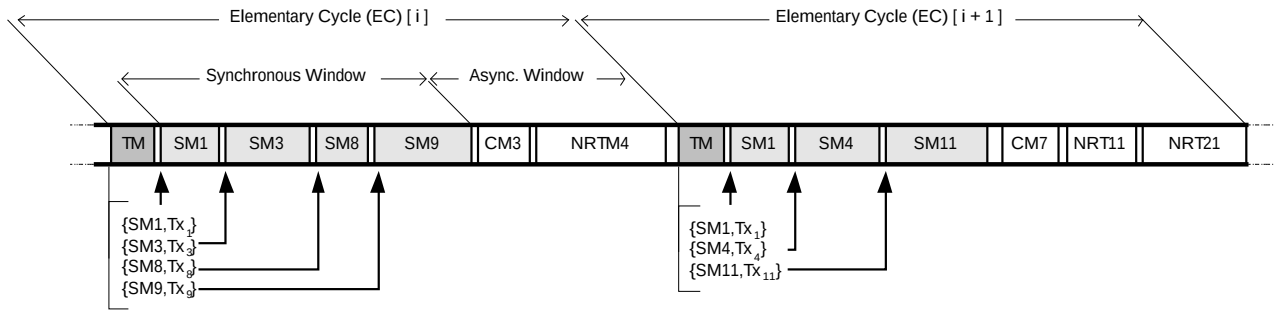

Figure 1. The Elementary Cycle structure.

In this case, SID_i is the node's identifier, $MaxDLC_i$ is the data length, in bytes, of the longest non-real-time message transmitted by that node, $MaxC_i$ is the respective maximum transmission time (including all overheads) and Pr_i is the node's non-real-time priority. 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, like the bus transmission speed, duration of the elementary cycle, minimum amount of bandwidth allocated to non-real-time traffic, protocol overheads dependent on the network topology (network length and number of hubs/repeaters), etc.

3.3. FTT-Ethernet schedulability constraints

The FTT-Ethernet protocol allows dynamic changes to the message streams that are exchanged on the network. However, to preserve the timeliness of the system, all the change requests are subject to admission control. Only changes that result in schedulable sets, and therefore do not jeopardize the system timeliness, are accepted.

The properties of the FTT-Ethernet protocol have impact on the schedulability analysis performed by the online admission control. Relevant factors are: time granularity, the transmission of a trigger message at the beginning of each EC and the enforcement of temporal isolation between synchronous and asynchronous traffic. All the time-related properties of the message streams, like period, deadline and initial phase, are expressed in multiples of the EC duration. Therefore, messages become ready synchronously, at the beginning of each EC, resulting in the absence of preemption instants.

The transmission of a trigger message at the beginning of each EC represents a protocol overhead, since it does not carry application related data. The length of this message depends on the maximum number of messages that are supported on a single EC. This parameter is fixed and set at pre-runtime. The length of the trigger message (LTM) is given by the following expression:

$LTM = max(ETH_{overhead} + FTT_{header} + 3 * MaxECmesgs\,; MinEthFrame)$

where: $ETH_{overhead}$ is the Ethernet frame overhead;
$FTT_{header}$is the FTT-Ethernet frame header contents;
$MaxECmesgs$ is the maximum number of messages allowed in each EC;
$MinEthFrame$ is the minimum length of an Ethernet frame.

The enforcement of temporal isolation between synchronous and asynchronous traffic is achieved by preventing the transmission of messages that do not fit within the respective window. Since the length of the messages is not correlated to the duration of the respective window, some idle time can appear at the end of each window, even when messages are waiting for transmission (inserted idle-time). The maximum amount of idle time in each window is given by:

$IIT_{Max} = max(C_i),\ i = 1..N$

where C_i is the transmission time of message i and N is the number of messages belonging to the window in consideration.

The scheduling model of the synchronous traffic in FTT-Ethernet, considering inserted idle-time and coarse resolution to express message parameters, follows closely the non-preemptive blocking-free model presented in [12]. In that work, several methods are suggested to adapt existing schedulability analysis for independent preemptive tasks with fixed priorities (e.g. [8] and [11]) to such model. Such adaptation can be further enlarged to apply directly to FTT-Ethernet when using fixed priorities-based scheduling. It suffices to consider that the remaining parts of the EC outside the maximum synchronous window (LEC-LSW) are included in the inserted idle-time (fig. 1). For dynamic priorities, such as when using EDF scheduling, a straightforward adaptation of existing schedulability analysis for independent preemptive tasks consists on considering the inserted idle-time as an extra virtual message with period of 1 EC [13].

Finally, in order to correctly calculate the transmission time of each message (equivalent to the execution time of a task) a correct characterization of the communication overheads per message transmission/reception must be carried out. These overheads are due to extra time (guarding time) that must be allocated between consecutive transmissions to guarantee that no collisions occur at the network access level. The necessity of such guarding time arises from variations in the instants of message reception caused by propagation delays in the network as well as from variations in the

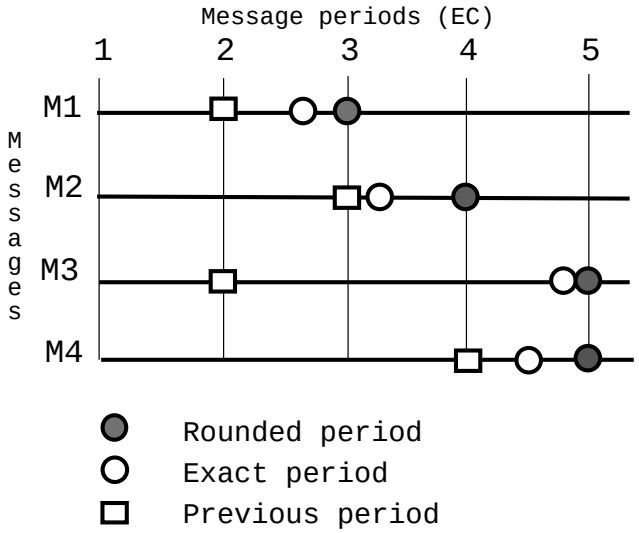

Figure 2. Rounding of periods in FTT-Ethernet.

transmission/reception instants of messages caused by variable latencies in the node's hardware and operating system. The propagation delay can be upper-bounded by knowing the transmission bit-rate and network topology. Concerning the frame transmission/reception jitter, an experimental characterization before system set-up must be carried out.

4. Applying the Elastic Task Model to message scheduling

The Elastic Task Model was originally developed for task scheduling in single microprocessors. Under this framework, tasks are considered preemptible. However, in the context of message scheduling, message transmissions cannot be suspended and resumed later, therefore preemption is not allowed. Another difference refers to the resolution used to express periods, initial phasings and deadlines. FTT-Ethernet uses a coarse resolution equal to the EC duration while in the original elastic task model the resolution can be arbitrarily small. Moreover, the transmission time of messages in FTT-Ethernet is always much smaller than the EC duration while in the elastic task model the task execution times are not constrained beyond a limited utilization factor.

Despite these differences, the elastic task model can be easily applied to FTT-Ethernet. With the simple adaptation suggested at the end of Section 3.3, equation 3 can still be used to generate new message utilizations (U_i). However, these utilizations do not necessarily lead to periods that are multiples of the EC duration (LEC) and thus, they must be rounded up (fig. 2) to the next integer multiple of LEC (T_i'), as in (10). The rounding must be done in excess, in order to guarantee that the resulting message set does not have a greater utilization factor than desired (U_d). After rounding up the periods, each message utilization U_i' is given by (11) and the overall effective utilization U'_{eff} is obtained by summing U_i' for all i. Due to the rounding ups of the periods, $U'_{eff} \leq U_d$ (fig. 3).

To avoid this situation and improve the efficiency of the

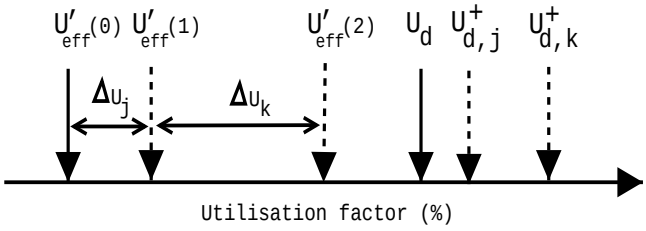

Figure 3. Increasing the effective utilization factor in FTT-Ethernet.

FTT-Ethernet implementation, the elastic task model was extended with an additional optimization step, performed after the initial compression algorithm, in which the slack utilization factor is better distributed among the messages. This redistribution is carried out coherently with the philosophy of the elastic model, guaranteeing that the resulting effective utilization factor does not exceed U_d (fig. 3).

The optimization step allows calculating a succession of effective utilization values $U'_{eff}(n)$ starting from U'_{eff} defined as above. Firstly, the process computes a vector with utilization values $U^+_{d,i}$ for every message i that can be decompressed and has utilization lower than the one resulting from equation 3 (Γ_ν), using equation 14. Each of these values corresponds to the increased overall utilization that would result if the utilization of message i was enlarged as in (12), due to reducing the respective period to the nearest integer multiple of LEC. The vector $\{U^+_{d,i}\}$ is sorted in ascending order and for each i, if $U'_{eff}(n) + \Delta U_i \leq U_d$ then $U'_{eff}(n+1) = U'_{eff}(n) + \Delta U_i$ and the period of message i is reduced by the duration of one EC (LEC). After scanning the whole vector, the final message periods impose an overall bandwidth utilization factor that is potentially closer to the desired value U_d.

$$\forall \tau_i \in \Gamma_v \quad T_i' = \lceil T_i \rceil = \lceil \frac{C_i}{U_i * LEC} \rceil * LEC \geq T_i \tag{10}$$

$$U_i' = \frac{C_i}{T_i'} \tag{11}$$

$$U_i^+ = \frac{C_i}{(T_i' - LEC)} \tag{12}$$

$$\Delta U_i = U_i^+ - U_i' \tag{13}$$

$$\forall \tau_i \in \Gamma_v \quad U^+_{d,i} = U_d + (U_i^+ - U_i)\frac{E_v}{E_i} \tag{14}$$

5. Experimental results

The Elastic Task Model has been implemented on the top of the S.Ha.R.K. kernel [14] with the FTT-Ethernet as the real-time communication protocol. A set of experiments on a multimedia application where performed. The same set of experiments was carried out also with Hub and Switch based Ethernet to assess the benefits of the presence of a deterministic communication layer.

The application developed consisted in the simulation of a video surveillance security system, containing a set of physically distant video cameras and a central console. Each camera can be served by distinct QoS, according to the current

$Cam.$	$C_i(FTT/ET)$	T_{i_0}	$T_{i_{min}}$	$T_{i_{max}}$	E_i
1	0.89/0.84	10	5	30	1
2	0.89/0.84	10	5	30	2
3	0.89/0.84	10	5	30	4
4	0.89/0.84	10	5	30	6

Table 1. Task set parameters used in the experiments.

(Periods and transmission times in milliseconds)

$Camera$	$t \leq 2s$	$2s < t \leq 5s$	$t > 5s$
1	10	5	10
2	10	10	10
3	10	15	10
4	10	20	10

Table 2. Periods of each message (ms) during the experiments.

bandwidth availability and the relevance of the data being sent. If enough bandwidth is available, all the cameras are allowed to sent frames at the nominal rate. Conversely, if the demanded bandwidth cannot be granted, higher bandwidth is reserved to cameras that transmit more relevant data.

The elastic guarantee mechanism has been implemented in the FTT Master node, acting both as QoS and admission control manager. All the change requests submitted to the Synchronous Messaging System are firstly submitted to the elastic guarantee mechanism. If the request results in an infeasible message set, it is rejected. Conversely, if the resulting message set is schedulable, the QoS manager calculates the new periods and updates the Synchronous Requirements Table. Since the SRT is used both by the QoS manager and the Scheduler, a mutex was used in order to guarantee data access consistency. Therefore, the updates to the SRT are atomic.

5.1. Experimental set-up description

The experimental set-up consists on 6 PC's, one acting as FTT Master, four as slaves, each producing a message stream associated to one camera, and, finally one PC dedicated to collecting network traffic data. The communication infrastructure was Ethernet at 10Mbps.

The simulated cameras have a resolution of 384*288 pixels and a color depth of 8 bits, yielding a frame size of 884.7 Kbit. The camera data frames are sent without any kind of compression. Since the image frame size is larger that the maximum Ethernet packet size, each image frame is split in 1000 Byte packets. A header containing the camera ID, frame and packet number, and packet data size is added to each packet, yielding a total Ethernet packet data size of 1010 Bytes.

The task set parameters used in the experiment are shown in Table 1, where C_i represents the message transmission time (@10Mbps) both for the FTT and Ethernet case, $T_{i_0}, T_{i_{min}}$ and $T_{i_{max}}$ are the nominal, minimum and maximum periods respectively and e_i is the message's elastic coefficient.

At the beginning of the experiment all cameras sent data at the nominal rate. At time $t = 2s$ camera 1 requested an increase in its QoS. This request is found to be feasible by the elastic guarantee mechanism as long as cameras 3 and 4 increase their transmission periods. The elastic task model found a feasible set with $\{T_1 = 5ms\,;\, T_2 = 10ms\,;\, T_3 = 15ms\,;\, T_4 = 20ms\}$. At time $t = 5s$, the QoS requirement

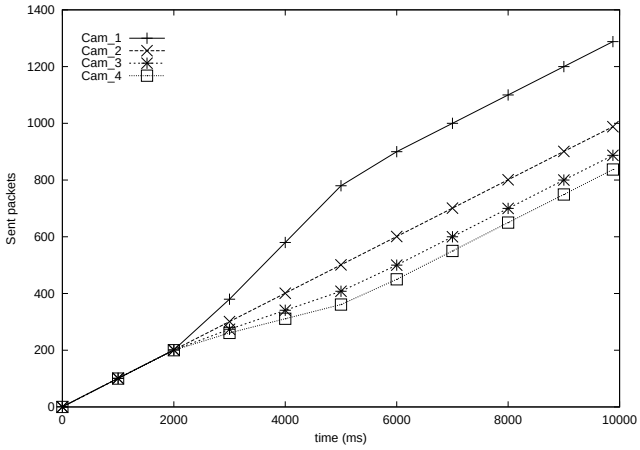

Figure 4. packets sent using FTT-Ethernet.

of camera 1 is reset to its nominal value, causing all the cameras to return to their nominal QoS.

The resulting message periods during the experiments are summarized in Table 2.

Practical experiments with this traffic pattern where made using both FTT-Ethernet and Hub and Switch based Ethernet.

5.2. Results with FTT-Ethernet

In the FTT-Ethernet setup the EC duration was set to 5ms (LEC=5ms) and the synchronous window was upper bounded to 37% of the EC (LSW=1.85ms), representing a maximum bandwidth of 3.7Mbps available for the synchronous traffic (SMS). This type of traffic was scheduled according to the EDF policy.

As referred at the end of Section 3, it is important to characterize and bound the communication overheads per message transmission/reception and include them in each message transmission time, for admission control and scheduling purposes. These overheads depend on both network properties, such as length and number of hubs, as well as on variable latencies imposed by the node's hardware and operating system in the transmission and reception of messages. The combined effect of these aspects was experimentally measured and upper bounded to 50 μs. Furthermore, each synchronous message also includes a specific FTT-Ethernet header [10] with additional control bytes. The resulting packet size, for 1000 data bytes, is 8896 bits resulting in a transmission time of approximately 0.890ms at 10 Mbps.

Figure 4 presents the number of packets transmitted by each of the nodes as a function of time, during the exper-

$Camera\ ID$	1	2	3	4
Rel. release jitter (avg) (%)	0.53	0.45	1.85	2.83
Absolute release jitter(%)	8.66	7.80	9.79	21.39

Table 3. message jitter with FTT-Ethernet.

$Camera\ ID$	1	2	3	4
Rel. release jitter (avg) (%)	0.66	1.71	1.13	0.69
Absolute release jitter(%)	66.44	91.65	90.33	90.81
Lost packets (%)	1.65%			

Table 4. message jitter (shared Ethernet).

iment. Initially, all cameras send packets at the same rate. However, at time $t = 2s$, the accumulated number of packets sent by each camera starts to diverge as a consequence of a request from camera 1 to increase its QoS. The elastic mechanism finds a feasible set, which results in an increase of the bandwidth assigned to this camera and a decrease in the bandwidth assigned to cameras 3 and 4. At $t = 5s$, camera 1 requests a QoS reduction to its nominal value. This implicitly causes the QoS of the remaining cameras to be increased to their nominal value, too. Consequently, from that moment on, all cameras start sending packets at the same rate again.

Table 3 summarizes the figures concerning the jitter suffered by the messages sent by each of the cameras. The values are presented in percentage and normalized to the respective message period. These values are relatively small despite the occurrence of changes in the message set, due to the control of transmission instants preventing the occurrence of collisions at the network access level.

5.3. Results with hub-based Ethernet

A second experiment was carried out using the same communication infrastructure as in the previous section, but without the use of the FTT-Ethernet layer. In each node a task was configured to reproduce the same data rate described above, at approximately the same instants, but without synchronization.

In this scenario, the Ethernet packet is composed of the data bytes plus a header, 10 bytes long, conveying information required to allow the consumers to identify and reassemble the data. The total packet size amounted to 8384 bits, corresponding to a transmission time of approximately 0.84 ms.

The number of packets sent by each node during the experiment follows a pattern very similar to the one obtained with FTT-Ethernet (fig. 4). However, as can it be observed in Table 4, there are, now, lost packets and an absolute release jitter that is considerably greater than the one experienced in the previous case.

It is interesting to observe that, despite using a relatively light load (around 35%), the event-triggered nature used in this approach leads to situations where, at some instants, several messages become ready simultaneously, originating collisions. In turn, these collisions result in a strong increase in the jitter figures and sometimes to lost packets.

$Camera\ ID$	1	2	3	4
Rel. release jitter (avg) (%)	6.13	0.32	11.00	17.01
Absolute release jitter (%)	66.61	74.61	83.30	126.41

Table 5. message jitter (switched Ethernet).

5.4. Results with switched Ethernet

In this case, the experimental setup is similar to the one described in the previous section, except that a switch was used to interconnect the nodes, instead of a hub. Again, the number of packets sent by each node during the experiment follows roughly the same pattern as in both previous cases. However, when comparing with the results obtained in the hub-based experiment, there are no frame drops, now. This result was expected, since the use of a switch avoids collisions, implements queuing at the switch ports and the total bandwidth requested was well below the network maximum throughput.

Concerning the jitter figures, shown in Table 5, it can be observed that the values for camera 4 are the greatest among all the experiments, with some messages delayed by more than one period. This phenomenon is explained by the buffering made at the switch ports.

6. Conclusions

In this paper was presented the application of the Elastic Task Model to message scheduling on a communication network using the FTT-Ethernet real-time communication protocol. The elastic task model was integrated in the FTT-Ethernet protocol, acting both as QoS and admission control manager, providing a framework in which periodic messages can be served by distinct QoS during system's normal operation. This model is particularly useful for distributed systems supporting dynamic environments, in which applications have to adapt to the varying operative conditions, leading to variations both in internal computational activities and messages exchanged by the underlying communication system. The policy for selecting a solution during run-time is implicitly encoded in elastic coefficients provided by the user at system configuration time.

A set of experiments was carried out, in order to assess the effectiveness of the proposed approach. The obtained results have shown that the architecture presented in this paper provides flexible communication, capable of handling dynamic sets of periodic messages, without jeopardizing the systems timeliness. Moreover, the same set of experiments was carried out over hub and switched-based Ethernet. Despite of the fact that no packet loss has occurred with the use of the switch, both of them have proved to be incapable of effectively handle time-constrained real-time messages, since large values of jitter were experienced.

References

[1] Thomesse, J-P, "Fieldbus and Interoperability", Control Engeneering Practice, 7(1), pp81-94, 1999.

[2] G. Beccari, S. Caselli, M. Reggiani, F. Zanichelli, ”Rate Modulation of Soft Real-Time Tasks in Autonomous Robot Control Systems”, IEEE Proceedings of the 11th Euromicro Conference on Real-Time Systems, York, June 1999.

[3] G. Buttazzo, G. Lipari, and L. Abeni, ”Elastic Task Model for Adaptive Rate Control”, Proceedings of the IEEE Real-Time Systems Symposium, Madrid, Spain, pp. 286-295, December 1998.

[4] G. Buttazzo and L. Abeni, “Adaptive Rate Control through Elastic Scheduling”, Proceedings of the 39th IEEE Conference on Decisionand Control, Sydney, Australia, December 2000.

[5] G. Buttazzo, G. Lipari, M. Caccamo, L. Abeni, “Elastic Scheduling for Flexible Workload Management”, IEEE Transactions on Computers, Vol. 51, No. 3, pp. 289-302, March 2002.

[6] T.-W. Kuo and A. K, Mok, “Load Adjustment in Adaptive Real-Time Systems”, Proceedings of the 12th IEEE Real-Time Systems Symposium, December 1991.

[7] C. Lee, R. Rajkumar, and C. Mercer, ”Experiences with Processor Reservation and Dynamic QOS in Real-Time Mach”, Proceedings of Multimedia Japan 96, April 1996.

[8] C.L. Liu and J.W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard real-Time Environment”, Journal of the ACM, 20(1), 1973, pp. 40–61.

[9] T. Nakajima, “Resource Reservation for Adaptive QOS Mapping in Real-Time Mach”, Sixth International Workshop on Parallel and Distributed Real-Time Systems, April 1998.

[10] Pedreiras, P., L. Almeida and P. Gai, “The FTT-Ethernet protocol: merging flexibility, timeliness and efficiency”, Paper accepted for publication at the Euromicro Conference on Real-Time Systems 2002, Vienna, June, 2002.

[11] Tindell, K., A. Burns, J. Wellings, “Analysis of Hard Real-Time Communications”, The Journal of Real-Time Systems, Vol.9, 147-171, 1995.

[12] Almeida, L., J. Fonseca, "Analysis of a Simple Model for Non-Preemptive Blocking-Free Scheduling", ECRTS01, 13th Euromicro Conference on Real-Time Systems, Delft, The Netherlands, 13-15 June 2001.

[13] Almeida, L., P. Pedreiras, J. Fonseca, “The FTT-CAN protocol: Why and How”. IEEE Transactions on Industrial Electronics, (to appear) December 2002.

[14] Paolo Gai, Massimiliano Giorgi, Luca Abeni and Giorgio Buttazzo, “A New Kernel Approach for Modular Real-Time Systems Development”, Proceedings of the 13th IEEE Euromicro Conference on Real-Time Systems June 2001 Delft, The Netherlands.