Improving the Real-Time Behavior of Ethernet Networks Using Traffic Smoothing

Lucia Lo Bello, *Member, IEEE*, Giordano Antonio Kaczyński, and Orazio Mirabella

***Abstract*—In modern process control systems, Ethernet is achieving a leading position, proposing itself as a network capable of supporting all communication needs at all levels in the Computer Integrated Manufacturing hierarchy. The main obstacle to using Ethernet at the Field level is the nondeterminism of the Ethernet MAC protocol, which cannot provide real-time traffic with bounded channel access times. This paper focuses on industrial applications featuring *soft* real-time constraints, such as periodic control or industrial multimedia, which do not require deterministic guarantees on deadline meeting. To cope with this class of applications, Ethernet should be able to guarantee the timely delivery of real-time packets in statistical terms.**

The paper presents *fuzzy* traffic smoothing, a technique to perform *adaptive* traffic smoothing over Ethernet networks at the Field level thus enabling them to provide a *statistical* bound on packet delivery time. Previous work showed that the fuzzy smoother outperforms other adaptive smoothers proposed in the literature. This paper addresses fuzzy smoother optimization through genetic algorithms. The proposed optimization is applied to tune the inference engine membership functions. The results obtained show the effectiveness of the approach.

***Index Terms*—Ethernet, fuzzy control, genetic algorithms, real-time communication, traffic smoothing.**

I. Introduction

MODERN process control systems create a hierarchy of plant activities. This hierarchy is reflected in the communication architecture, which features three levels with different traffic requirements. The use of different protocols at the various levels represents a limit, and complicates the information exchange between networks at different levels. Both manufacturers and users are therefore making great efforts toward harmonizing the communication infrastructure at the plant level. In this context, Ethernet is achieving a leading position, proposing itself as a network capable of supporting all communication needs at all levels in the control hierarchy [1]. Ethernet is gaining ground over other technologies at the plant Backbone level, and it is widely used at the Cell level. However, at the Field level, where real-time constraints are imposed on communication, the nondeterminism of the Ethernet MAC protocol [2], which cannot guarantee that data delivery deadlines will be met, represents a great obstacle.

This paper focuses on industrial applications featuring *soft* real-time constraints which do not require deterministic guarantees on deadline meeting and can be satisfied with a *statistical* bound on packet delivery time. Statistical guarantees on deadline meeting satisfy the requirements of real-time non-mission-critical applications, such as periodic control or industrial multimedia, which can tolerate a small violation probability with delay bounds. In an automated manufacturing system, for example, real-time periodic control messages may need to be delivered within 20 ms of their generation with a 98% probability, while voice packets may require delivery within 40 ms with a 94% probability, and so on. To cope with this class of soft real-time applications, Ethernet should be able to guarantee the timely delivery of real-time packets in statistical terms, i.e., with a given probability. To this end, *traffic smoothing* [3] can be used at the Field level. Traffic smoothing is a technique which enables an Ethernet to support soft real-time communications requiring a *statistical* bound on packet delivery time.

This paper deals with fuzzy traffic smoothing, a soft computing-based technique to perform adaptive traffic smoothing over Ethernet networks and addresses optimization of the fuzzy smoother through genetic algorithms. The paper is organized as follows. After discussing the case for traffic smoothing over Ethernet networks, Section II focuses on *fuzzy* smoothing and compares its performance with that of another adaptive smoother proposed in the literature. Section III addresses optimization of the fuzzy traffic smoother through genetic algorithms, while Section IV presents and discusses the experimental results obtained. Finally, Section V gives our concluding remarks.

II. Traffic Smoothing Over Ethernet Networks

When both real-time (RT) and non-real-time (NRT) packets are transported over an Ethernet, RT packets from a node may experience a long delay due to a) contention with NRT packets in the source node and b) collision with RT and NRT packets from the other nodes. In [3], Shin *et al.* analytically demonstrated that, to statistically bound the medium access time for an Ethernet frame, it is sufficient to keep the total arrival rate for new packets generated by stations below a threshold called the *network-wide input limit*. This makes it possible to achieve statistical real-time communication over an Ethernet. However, the Ethernet MAC protocol is totally distributed, and no station is aware of either the current packet arrival rate for the whole network or the transmission needs of other nodes. As a result, the only way to enforce the network-wide input limit on an Ethernet is to enforce the limit on a per-station basis. In [3], each station is assigned a local threshold, called a *station input limit*, and a middleware called a *traffic smoother* is implemented on each node to regulate the outgoing NRT stream, to maintain the traffic generation rate below the station input limit. Several traffic smoothers have been proposed in the literature, for either Shared [3]–[6] or

Manuscript received May 24, 2004; revised December 5, 2004.

The authors are with the Dipartimento di Ingegneria Informatica e delle Telecomunicazioni, University of Catania, Catania, Italy (e-mail: lucia.lobello@diit.unict.it; giordano.kaczynski@wp.pl; omirabel@diit.unict.it)

Digital Object Identifier 10.1109/TII.2005.852071

Switched Ethernet [7], which differ in the way the station input limit is enforced. The traffic smoother in [3]–[6] is implemented at a kernel level, as a software layer inserted between the TCP/IP (or UDP/IP) and the Data Link layer. Within a node, RT packets are distinguished from NRT packets using the Type Of Service (TOS) field in the IP header, and a priority queue with two priority levels, high for RT and low for NRT packets, is maintained. An RT packet is not affected by smoothing, while NRT traffic is transmitted as long as the overall station arrival rate (which comprises both RT and NRT packets) is below the station input limit; otherwise NRT packets are delayed. The traffic smoother therefore has two main effects: first it gives RT packets priority over NRT ones, in order to eliminate contention within each local node, and secondly it smoothes NRT traffic so as to reduce the likelihood of collision with RT packets from the other nodes. As a result, the number of packet collisions on the network is dramatically decreased.

Implementation of the traffic smoother only requires a minimal modification of the kernel, i.e., in the device driver for Linux (or a new network driver interface specification (NDIS) [8] for Windows NT), and does not entail any changes to the current standard Ethernet MAC protocol or TCP/IP (or UDP/IP) stack. What has to be modified is the Ethernet device driver to record the time when a packet in the Network Interface Card (NIC) experiences a collision, so that the smoothing algorithm may use it. Traffic smoothing is based on a *credit bucket* mechanism, which is a *leaky bucket*-based algorithm [9]. The credit bucket has two parameters: Credit Bucket Depth (CBD), which indicates the capacity of the credit bucket, and Refresh Period, which indicates the replenishment period (RP). Up to CBD credits are added to the bucket every RP seconds. CBD limits the maximum number of credits that can be stored in the credit bucket. If the number of credits exceeds CBD, overflow credits are discarded. When a packet arrives at the traffic smoother, if there is at least one credit in the bucket, the traffic smoother forward it to the Ethernet NIC and removes as many credits as the size of the packet (in bytes). When the number of available credits is lower than the packet size, credits are allowed to be borrowed from the next credit budget. So, the balance of credits can be negative until the next replenishment, when an algebraic sum of the credits in the bucket will be performed. If, on the other hand, the number of credits in the bucket when a packet arrives is less than or equal to zero, the packet is not transmitted to the Ethernet NIC until at least one credit becomes available following a replenishment. The CBD/RP ratio is the station input limit. The solution proposed in [3] was *static* traffic smoothing, where each station is assigned a given, constant, station input limit. Static smoothing is easy to implement but it suffers from two main problems, i.e., inflexibility and lack of scalability. To overcome these limitations, adaptive traffic smoothing was proposed, which allows a station to dynamically modify the station input limit it is assigned, according to the current network workload. For this to be possible, it is necessary to know the workload on the network at any one time. The problem is solved by activating a suitable user process in each machine (called a *sniffer*) to monitor the global traffic trends. This is possible because in Ethernet-based networks a transmitted frame is listened to by all the other stations. In order to evaluate the current network workload and, therefore, to guide adaptive smoothing, different approaches have been used, based on the measurement of either throughput [5] or the number of collisions. The *Harmonic-Increase and Multiplicative-Decrease* (HIMD) approach described in [4] applies an adaptation mechanism which reacts to the detection of a single collision over a given time α. When a collision is detected, the RP is increased by whichever is lower between twice its current value and a given $\mathrm{RP}_{\max}$ value, while in the absence of collisions the RP is periodically decreased (with a period of τ) by a constant Δ heuristically determined down to a value of $\mathrm{RP}_{\min}$. The parameters α, Δ, τ, $\mathrm{RP}_{\max}$ and $\mathrm{RP}_{\min}$ are user-controllable. By using different values, different delay and throughput characteristics can be obtained. The HIMD smoother suffers from some limitations, which motivated the development of another approach, the fuzzy traffic smoother.

A. Fuzzy Traffic Smoothing

The *fuzzy traffic smoother* [6] is an adaptive traffic smoother based on a fuzzy controller. It has two inputs—the number of collisions and the overall throughput observed in a reference interval—and a single output, here called *VarRP*, which represents the variation of the *refresh period* as compared with the current value. If RP_{old} is the current *refresh period* and RP_{new} is the new value, the formula used by the smoothing driver is

$$\mathrm{RP}_{new} = \mathrm{RP}_{old} + VarRP. \tag{2.1}$$

The fuzzy smoother represents an improvement on previous work on adaptive smoothing in two respects. First, it uses both total throughput and the number of collisions as input parameters for the smoother, and these two parameters together give a more complete picture of the actual workload than the detection of a single collision over an interval of length α. One criticism that can be made of the HIMD approach [4] is that the occurrence of a single collision is not necessarily due to network congestion, but may derive from a temporary coincidence between the transmission requirements of two or more stations when the load on the network is not particularly high. According to the throughput value, therefore, doubling the RP value when a collision occurs may be too drastic or excessively penalizing for NRT traffic. The second improvement offered by the fuzzy smoother, as compared with previous dynamic smoothers, is that the variation of the station input limit is not based on fixed variations as in [4] and [5], but is dynamic and gauged according to the actual workload by the *fuzzy controller* which, according to the total throughput and number of collisions, applies rules to choose the most appropriate RP on a case-by-case basis.

Fuzzy control is particularly suitable when knowledge of the system to be controlled is insufficient or the dynamic model is too complex to model and control. This is the case of the system considered here, i.e., Ethernet combined with traffic smoothing, which, due to its nonlinear and quite complex behavior, is difficult to model and control using traditional controllers, as they rely on some knowledge of the model of the system to be controlled. The system here is, on the other hand, highly suitable for control that is capable of integrating heuristic knowledge acquired in the field by experts. Three membership functions were

TABLE I
INFERENCE RULES USED

If *Collisions* is	and *Throughput* is	then *VarRP* is
Low	*Low*	$-0.5 \cdot RP_{min}$
Low	*Medium*	$-0.3 \cdot RP_{min}$
Low	*High*	0
Medium	*Low*	$-0.1 \cdot RP_{min}$
Medium	*Medium*	0
Medium	*High*	$0.2 \cdot RP_{max}$
High	*Low*	RP_{max}
High	*Medium*	$0.7 \cdot RP_{max}$
High	*High*	$0.6 \cdot RP_{max}$

defined for each input to the fuzzy controller, corresponding respectively to the values (*low, med, high*) of each variable. This number was chosen heuristically to represent all the possible operating modes of the system in relation to the values taken by the input parameters, without introducing an excessive number of combinations (and thus inference rules) to be defined. As there are two input variables, each of which can have three different *membership functions*, there are $3^2 = 9$ combinations, i.e., nine rules. The membership functions chosen for the input variables were triangular and trapezoidal, as they are the most suitable for representing the type of inputs being examined. For the output variable, *singleton* membership functions were chosen, i.e., each set comprises a single point with a degree of membership of 1 (*crisp*). To complete the controller, the *inference rules* were defined, indicating the control action to apply according to all the possible combinations of the input variables. The structure of an inference rule in our case is

$$\text{If } Collisions \text{ IS } \langle Low \rangle \text{AND } Throughput \text{ IS} \langle Low \rangle \text{THEN } VarRP \text{ IS } \langle value \rangle. \quad (2.2)$$

In [6], the behavior of the fuzzy smoother was investigated in a real scenario, comparing its performance with that of the HIMD smoother. The experimental test-bed comprised 11 workstations equipped with Linux operating system and connected via a 10BASE-T Ethernet. The smoother was implemented as a Linux kernel module and the fuzzy algorithm via the Fuzzy Studio[1] 2.0 program [10] developed by SGS-Thomson Microelectronics. The *fuzzy* controller comprised the *inference rules* shown in Table I. The KURT (KU Real-Time) patch developed by the ITTC (Information and Telecommunication Technology Center) at the University of Kansas, which includes and extends UTIME [11], was used to increase the resolution of the timers, as Linux cannot handle timers with a resolution of less than 10 ms. Both the fuzzy and HIMD smoothers were implemented, and a performance comparison was made in the same environment and operating conditions as outlined in [4]. In both cases, the smoothing driver was activated on each node with an $\mathrm{RP}_{\max}$ of 100 ms, an $\mathrm{RP}_{\min}$ of 3 ms and a τ of 1 ms; the observation period for the sniffer process used to measure the network load was set to 10 ms. As in [4], we measured the *roundtrip delay* for RT control messages exchanged between two processes, called Client and Server, running on two different PCs, and calculated the deadline miss ratio for RT messages, taking the *deadline* for a complete transaction to be 129.6 ms. The duration of RT transactions was measured with a growing workload, progressively activating the processes generating NRT bursts, called Station processes. In [6], a complete description of the experimental scenario is given. Here, for the sake of brevity, we only show some results for comparative purposes. Fig. 1(a) and (c) show how throughput and roundtrip delay are distributed with a varying *workload* during bursts using HIMD smoothing. The five bursts are of increasing intensity, ranging from only one NRT Station active to five NRT Stations active. Whereas roundtrip delay values are quite low in the absence of bursts, they increase considerably during bursts and several messages miss their deadlines. It can be also observed that the delays affecting RT messages are high even between one burst and another. This is because the HIMD approach delays the handling of NRT traffic beyond the end of the burst, thus keeping the throughput high for a certain time after the burst. This is confirmed by the throughput curve. Fig. 1(c) shows that several RT messages feature high roundtrip delay values even after the NRT workload burst has ended, because a number of NRT messages have not been dispatched, so the overall traffic is still high. On the other hand, the roundtrip values for the single messages with varying workloads from all the processes, both RT and NRT, obtained using the fuzzy smoother and shown in Fig. 1(d), show that the delays are much shorter, even when there are bursts. Very few messages feature roundtrip delay values over the deadline. This means that only a few messages collide with NRT traffic. As all the NRT traffic is dispatched during bursts, RT messages outside burst periods are not affected by an appreciable delay. The improvement in performance achieved by fuzzy smoothing as compared with the HIMD approach is also confirmed by the total network throughput graph in Fig. 1(b), which shows a more regular trend and higher throughput values than the graph in Fig. 1(a). This is due to the fact that the fuzzy smoother does not totally block NRT traffic, which may reach values very close to the workload. Moreover, in the time interval between consecutive bursts there is no "tail" due to the previous burst, because all the traffic has been handled on time. Thus, unlike the HIMD case, the effect of the bursts does not spread out the burst period. Finally, a deadline miss ratio (D_{mr}) comparison between the two approaches showed that, with a deadline of 129.6 ms, the D_{mr} never exceeds 0.7% when the fuzzy smoother is used, which is about 1/3 of the value obtained using HIMD. This result is highly satisfactory for many soft RT applications.

III. FUZZY SMOOTHER OPTIMIZATION BASED ON GENETIC ALGORITHMS

The successful results obtained in [6] suggested looking for a way to further improve the performance of the fuzzy controller. *Genetic Algorithms* (GAs) were chosen to accomplish this task, as several works (e.g., [12]–[14]) have confirmed the efficiency of GAs in the design of a fuzzy controller to obtain desired behavior from the system being controlled. We therefore used GAs to tune the parameters of the membership functions of our fuzzy traffic smoother. There is no run-time overhead for

[1]Fuzzy Studio is a registered trademark of SGS-Thomson Microelectronics.

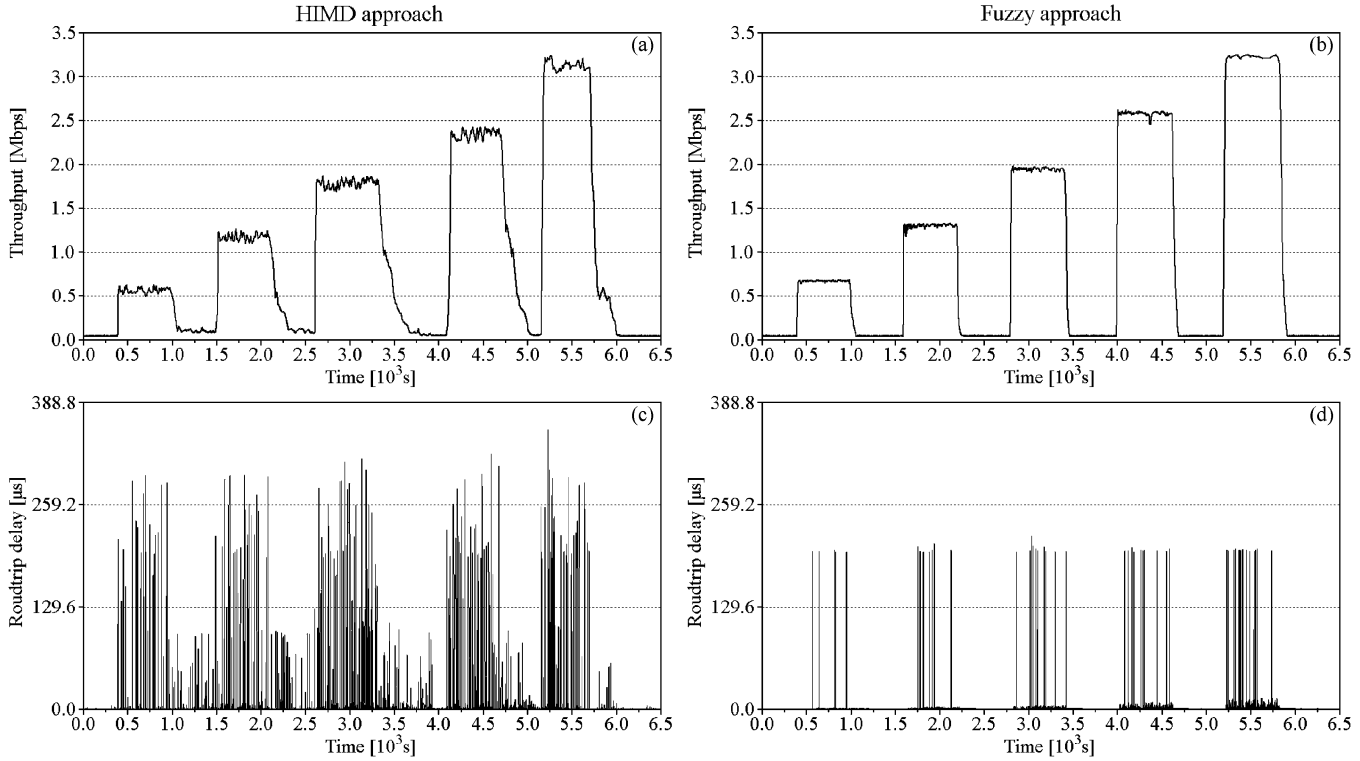

Fig. 1. Performance comparison between HIMD and fuzzy smoothing. (a) HIMD throughput. (b) Fuzzy smoother throughput. (c) HIMD roundtrip delay. (d) Fuzzy smoother roundtrip delay.

the genetic algorithm. It runs off-line, performing genetic optimization according to the requirements of the scenario being considered. In Sections III-A–G, the steps taken to optimize the fuzzy smoother are described and discussed.

A. Coding

It was decided to use standard binary coding for the chromosomes and to encode each smoother input (throughput and collisions) in 8-bit strings. A value of 8 bits was chosen because a lower number was deemed insufficient: it would, in fact, have entailed an excessively loose discretization of the search space. Higher values, on the other hand, would certainly have increased the accuracy of the search, but would have involved greater complexity. Tests carried out in with various values confirmed the 8-bit solution as being a successful trade-off. As the inputs fall within the real number intervals of [0,10] and [0,16] respectively, they have to be normalized in the range of integers that can be represented with 8 bits, i.e., [0,255]. Having chosen the type of coding and the range in which to normalize the inputs, it was necessary to choose the type of mapping to adopt. The following mapping function was chosen [14], [15]:

$$r = r_{\min} + \frac{b \cdot (r_{\max} - r_{\min})}{(2^m - 1)} \tag{3.1}$$

where r is the real value to be normalized, $r_{\max}$ and $r_{\min}$ are the upper and lower bounds of the range of real numbers, and b is the integer represented by the string of m bits.

In the reverse operation (going from a real number to a binary number) accuracy is often lost, so it is necessary to find an approximation that will not jeopardize the functioning of the GA. We chose to round down when the second decimal was less than or equal to 5 and to round up in other cases.

B. Structure of a Chromosome

The shapes of the membership functions were left the same as those of the original fuzzy smoother which has to be tuned. They are trapezoid for the linguistic terms High and Low, and triangular for the term Medium. Fig. 2 shows the parameters of the membership functions to be optimized. The sub chromosome for a generic i-th input will therefore be

$$C_i = (L_{1i}, L_{2i}, M_{1i}, M_{2i}, M_{3i}, H_{1i}, H_{2i}) \tag{3.2}$$

where the parameters must comply with the following constraints:

$$\begin{aligned} L_1 &< M_2 \\ L_1 &< L_2 \\ M_2 &< M_3 \\ M_2 &> M_1 \\ M_2 &< H_2 \\ H_2 &> H_1 \\ M_3 &\geq L_2 \\ H_1 &\geq M_1 \\ M_3 &\geq H_1 \\ L_2 &\geq M_1. \end{aligned} \tag{3.3}$$

The resulting chromosome, given that there are two inputs, will be

$$C_i = C_{1i} \cup C_{2i} \tag{3.4}$$

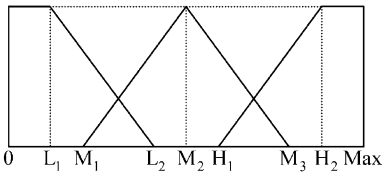

Fig. 2. Parameters of the membership functions to be optimized.

represented by $7 \times 8 \times 2 = 112$ bits. We recall that the smoother output is not involved in the structure of the chromosome because we are not interested in tuning it. The minimum and maximum values (0 for both inputs; 10 for throughput and 16 for collisions) are not encoded in the chromosome, as they are constant and are added in the decoding stage.

C. Initial Population

The number of individuals chosen for the population, after several tests, was 40 and did not change during the evolution. During the various simulations the 40 initial individuals were at times generated in a random manner, whereas at others one or more individuals were inserted ad hoc, using known configurations obtained from previous simulations or on the basis of acquired knowledge of the problem.

D. Fitness Function

As is known, the validity of a solution to an optimization problem dealt with by using GAs is determined by a function $f(x)$, called the *fitness* of the solution. The fitness function chosen does not refer directly to the fuzzy controller output, as happens in some applications of GAs to fuzzy logic, but to the effects a fuzzy controller has on the network in the context of a whole simulation. The fitness function assigned to a given chromosome x is

$$f(x) = 0.5 \cdot (100 - D_{mr} \cdot 100) + 0.5 \cdot (E_b \cdot 100) \qquad (3.5)$$

where D_{mr} is the percentage of RT frames, out of the total number generated, that miss their deadlines and E_b is the *input_burst_duration/output_burst_duration* ratio.

The fitness function takes values in the range [0,100]: the higher it is, the better the solution represented by the chromosome. The $f(x)$ chosen is an attempt to find the right tradeoff between the need to meet deadlines for RT frames and to end NRT bursts in as short a time as possible. This choice is not only dictated by the need to avoid sacrificing NRT traffic, but also because the shorter the bursts are, the lower the probability of overlapping bursts from different stations degrading the performance of RT traffic even further. In multi-objective optimizations the Weighted Sum Approach, i.e., the weighted sum of the objectives to be optimized, is commonly used [16], [17] so as to change the multi-objective problem into a scalar one. Knowing that a genuine genetic algorithm needs scalar fitness information to work, the simplest idea that we could devise was to combine all the objectives into a single one using addition, multiplication, or any other combination of arithmetical operations that we could think of. There are, however, obvious problems with this approach. The first is that we have to provide some accurate scalar information on the range of the objectives, to avoid having one of them dominate the others. This implies that we should know, to a certain extent, the behavior of each of the objective functions. This is not only the simplest approach, but also one of the most efficient, since it requires no further interaction with the fitness function: once chosen, the fitness function requires no further modification. If the GA succeeds in optimizing the resulting fitness function, then the results will be at least suboptimum in most cases. In our case, as the system model is not known and we do not have the transfer function, we acquired knowledge of the behavior of the objective functions through a number of experiments. After implementing the evolutionary process, we performed a sensitivity analysis of the weights of the objective function. The results obtained showed that the best tradeoff for the requirements of the system being considered was achieved with weights of 0.5 for both objectives. To assess the validity of the result obtained using the Weighted Sum Approach, we also performed a search for the Pareto front [16], [17]. More details are given in Appendix.

In the fitness values we used in our optimization, the values of D_{mr} and E_b were obtained from the outputs of a simulator we developed for optimization purposes, which is written in *Java*. The membership parameters were coded by a chromosome inserted into the fuzzy controller of the simulator and the average deadline miss ratio and burst duration values over ten simulations were used to calculate the fitness of the chromosome on the basis of function (3.5).

E. Stop Condition

In order to establish when to stop the evolutionary process, it was decided to set a limit on the number of generations. The limit chosen was 50 generations, a value beyond which no further improvement was achieved during the numerous optimizations performed using our reference scenarios, which will be depicted in Section IV.

F. Genetic Operators

The genetic operators used on each population to create the next one were as follows.

- — *Roulette wheel selection*: Selection is not completely random but linked, via randomly generated values, to the ratio between the fitness of the chromosome being investigated and the overall fitness of the population. Each chromosome is assigned a certain probability of selection, proportional to its relative fitness.
- — *Crossover*: Here we used one point crossover, which consists of obtaining two individuals from an original pair by defining a randomly chosen crossover point and exchanging the binary semi strings thus obtained. The crossover was applied with a single randomly chosen cutoff point, as a larger number gave no apparent advantages.
- — *Mutation*: This was applied by complementing a randomly chosen bit of the chromosome.
- — *Elitism*: To prevent the best chromosome in the generation from being lost via mutation and crossover, the best individual was copied directly into the population of the next generation before applying the other operators.

The crossover and mutation probabilities, p_c and p_m, are not initially set, but computed dynamically during the evolution. The formulae used were [14]:

$$p_c = \exp\left(\frac{-G}{M}\right) \text{ and } p_m = \exp\left(\frac{0.1 \cdot G}{M}\right) - 1 \qquad (3.6)$$

where G is the current generation and M the maximum one.

Initially, therefore, at the first generation $p_c = 1$, decreasing exponentially toward a value of 0.1 as the evolution proceeds. Mutation, on the other hand, is initially 0, increasing exponentially up to a final value of about 0.25. This value may seem excessively high, but given the large number of nonvalid chromosomes that can result from mutation, it is necessary to have a sufficiently high probability for mutation to produce a sufficient number of valid new solutions. We define a valid solution as one which does not violate the constraints on the membership function parameters defined in (3.3). Nonvalid chromosomes are eliminated immediately and thus are not part of the new population. In this way, we are certain that the population is always made up of valid chromosomes. It would have been possible to limit the genetic operators' mode of operation a priori, thus preventing them from generating mutating chromosomes, but this would have led to far greater complexity in implementing the operators. Initially, thanks to the high crossover probability, it is possible to have a high degree of recombination of the substrings in the search space, which is initially very large. Then, as the evolution proceeds, this probability decreases and mutation increases, i.e., the possibility of a refined exploration of the smaller search space concentrated around the optimal solution obtained. This method proved to be a good one, overcoming local maxima quickly and exploring a very large number of possible solutions to the problem. Selection was implemented as follows.

1) The fitness of each individual x_i, belonging to the set $\{x_1, x_2, \ldots, x_N\}$ of N chromosomes in the current generation, is modified on the basis of the following formula [15]:

$$f'(x_i) = \exp(f(x_i)). \qquad (3.7)$$

2) The total fitness of the population is calculated as:

$$F = \sum_{i=1}^{N} f'(x_i) \qquad (3.8)$$

where N is the number of individuals in the population.

3) The relative fitness of each individual x_i is calculated as:

$$f_r(x_i) = \frac{f'(x_i)}{F}. \qquad (3.9)$$

So the value $f_r(x)$ corresponds to the probability of individual x being selected, according to the classical *roulette wheel* method, with the addition that the fitness function has been modified by raising its power. This method is very useful when the differences between the fitness values are minimal [15], and proved to speed up the optimization process considerably, in terms of the number of epochs needed to reach a global maximum.

G. Structure of the Algorithm Used

The genetic algorithm was implemented according to the following steps.

1) Construction of an initial population of N individuals.
2) Evaluation of the fitness of each individual, obtained from the outcome of a simulation for each of them, with the same topology and workload.
3) Selection of the elite element and insertion into the new population.
4) Roulette wheel selection of a meta-population of $N - 1$ individuals.
5) Application to the meta-population of crossover and mutation on the basis of their probabilities.
6) Verification of the offspring obtained.
7) Substitution of parents in the meta-population with valid offspring.
8) Execution of steps 5 to 7 N times.
9) Turning the meta-population into the new one for the next generation.
10) Return to point 2 if the set number of epochs has not been reached.
11) End. The element with the greatest fitness is the result of the optimization.

As in any optimization with GAs, account is taken of the average fitness of the population during the evolutionary process to ensure it is proceeding correctly. It would, in fact, be wrong to refer to the element with the greatest fitness, which could be a "strange" case in a population of mediocre individuals. What is being sought is an optimally adjusted element in a group of well-adjusted individuals. This is even more true in our case, given the highly stochastic nature of an Ethernet.

IV. Performance Evaluation

The performance of the optimized fuzzy smoother was evaluated on a test-bed comprising ten workstations equipped with Linux operating systems and connected via a 10BASE-T Ethernet. The reference environment was a factory communication system with soft real-time requirements. All stations generate the same amount of soft RT traffic. The RT and NRT traffic were produced by two different generation processes implemented in the nodes. Two scenarios with different RT traffic deadlines were evaluated: the first one, henceforward called Scenario 1, features light RT traffic (100 Kbps) with a urgent deadline, i.e., 1 ms (thus emulating fast-dynamic control systems); the second one, henceforward called Scenario 2, features heavier RT traffic than the previous one (1.5 Mbps), with a less urgent deadline, i.e., 50 ms (thus emulating industrial multimedia applications). In both cases, while all the nodes generate RT traffic, there are 5 of them which also generate bursts of NRT traffic that heavily load the network.

The optimized membership functions of the fuzzy controller were obtained for each scenario. They are shown in Fig. 3, while Table II gives the optimized values obtained for the parameters of the membership functions. We can observe that the membership functions obtained for Scenario 1 are quite different from

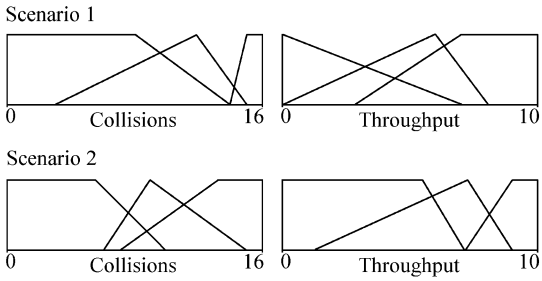

Fig. 3. Optimized membership functions for the two scenarios considered.

TABLE II
PARAMETERS OF THE OPTIMIZED MEMBERSHIP FUNCTIONS

Parameter	Scenario 1		Scenario 2	
	Collisions	*Throughput*	*Collisions*	*Throughput*
L_1	8.031	0	5.521	5.49
L_2	13.99	7.019	9.913	7.137
M_1	3.011	0	6.023	1.254
M_2	11.85	6	8.972	7.254
M_3	14.99	8.078	14.99	9.019
H_1	13.99	2.862	7.09	7.176
H_2	14.99	7.019	13.23	9.019

those obtained for Scenario 2, due to the different deadline and RT traffic workload values used during the optimization process (while the NRT traffic burst was 5 Mbps in both cases). In Scenario 1, the RT workload is low, but the deadline is urgent. For this reason, the optimized Throughput function is more severe than the one obtained in Scenario 2, while the membership function for Collisions is less severe than in Scenario 2. This is because in Scenario 2 the overall throughput is higher than in Scenario 1, so in order not to penalize NRT traffic, the Throughput membership function has to be less severe, while RT performance is enforced by a more severe membership function for Collisions.

In the following, performance figures will be shown. All the graphs refer to measures taken during 300 seconds of operation. For reasons of graphics, the values measured were averaged over 50 ms windows; single peaks in the graphs are smoothed in the observation window being considered. On the x-axis, time is expressed as the sequence number of the observation windows.

Scenario 1: In this scenario the ten stations produce a total RT workload of 100 Kbps with 1-Kbit frames. We first evaluated the number of collisions occurring in the various operating conditions. Two set of measurements were performed, with and without smoothing, using two different types of burst, henceforward referred to as Burst A and Burst B.

Burst A: 5 stations generating an NRT burst of 6.8 Mbps lasting 60 s.

Burst B: 5 stations generating an NRT burst of 11 Mbps lasting 60 s.

Fig. 4(a) shows the number of collisions during a burst A of NRT traffic without smoothing. If we increase the workload up to the network saturation point (with a type B burst), we see that the number of collisions increases and the overload situation is prolonged, as shown in Fig. 4(b). This is due to the limited capacity of the network, which remains congested even after the burst. The effect of smoothing on collisions is very interesting because it shows the smoother's capacity to serve all the traffic (even NRT) efficiently. As Fig. 4(c) shows, in the same workload conditions as Fig. 4(a) (Burst A), the number of collisions is drastically reduced. The peak that can be observed at the start of the burst depends on the response time of the smoother, which takes some tens of milliseconds to apply its control action. Fig. 4(d) shows the number of collisions obtained in the same scenario as in Fig. 4(b) (Burst B) with smoothing. As can be seen, the smoother reduces the number of collisions, but this time the overload situation lasts longer than the NRT burst, and is greater than when smoothing is not applied. This is because the need to maintain the performance levels required for RT traffic means limiting NRT throughput, so the burst takes longer to end than it does in the absence of smoothing.

The second set of measurements refers to throughput and delay. Graphs in Fig. 5, which show the overall throughput with and without smoothing, reveal that the fuzzy smoother also improves throughput for NRT traffic. This positive effect on the network's capacity to serve NRT traffic derives from the drastic reduction of the number of collisions obtained by fuzzy smoothing. This benefit to NRT traffic is a very interesting result, since the function of smoothing is above all to privilege RT traffic. On the other hand, when no smoothing is applied, the throughput burst is prolonged: due to the high number of collisions, the traffic is occasionally blocked and thus takes longer to serve. As far as RT traffic delivery delay is concerned, we first measured the transmission delay (i.e., the time between the generation of a message and its delivery to the remote station) without bursts and compared it with the deadline (for soft RT applications, it is not very important to know the actual delay for single messages but the percentage of deadline miss). The results given in Table III show that without bursts there were no deadline misses for RT traffic. We recall that the value obtained depends not only on the time required to send a 1-Kbit frame, but also on the various delays introduced by the operating system (due to the presence of other processes that are needed for acquisition of the information required by the smoother), the Transport layer (UDP) and the hub. Then, the effect of Burst A and Burst B on RT traffic delivery delay was assessed. With Burst A, in the presence of smoothing [Fig. 6(a)], the RT delay increases to a limited extent and only during the workload burst. Without smoothing [Fig. 6(b)], the delay is much greater and is prolonged beyond the workload burst, because the overloaded network needs more time to end the burst. The same measurements were repeated for Burst B. In this case, the smoother behaves differently, because the very high workload (beyond network saturation) forces the smoother to penalize NRT traffic. We observed that the smoother prolongs the transmission burst (if compared with the duration of the workload burst) due to the need to limit the number of collisions. Table III shows that the number of deadline misses is significantly reduced by smoothing in the presence of both type of bursts. The positive effect of the smoother on both RT and NRT traffic is due to the optimization performed on the

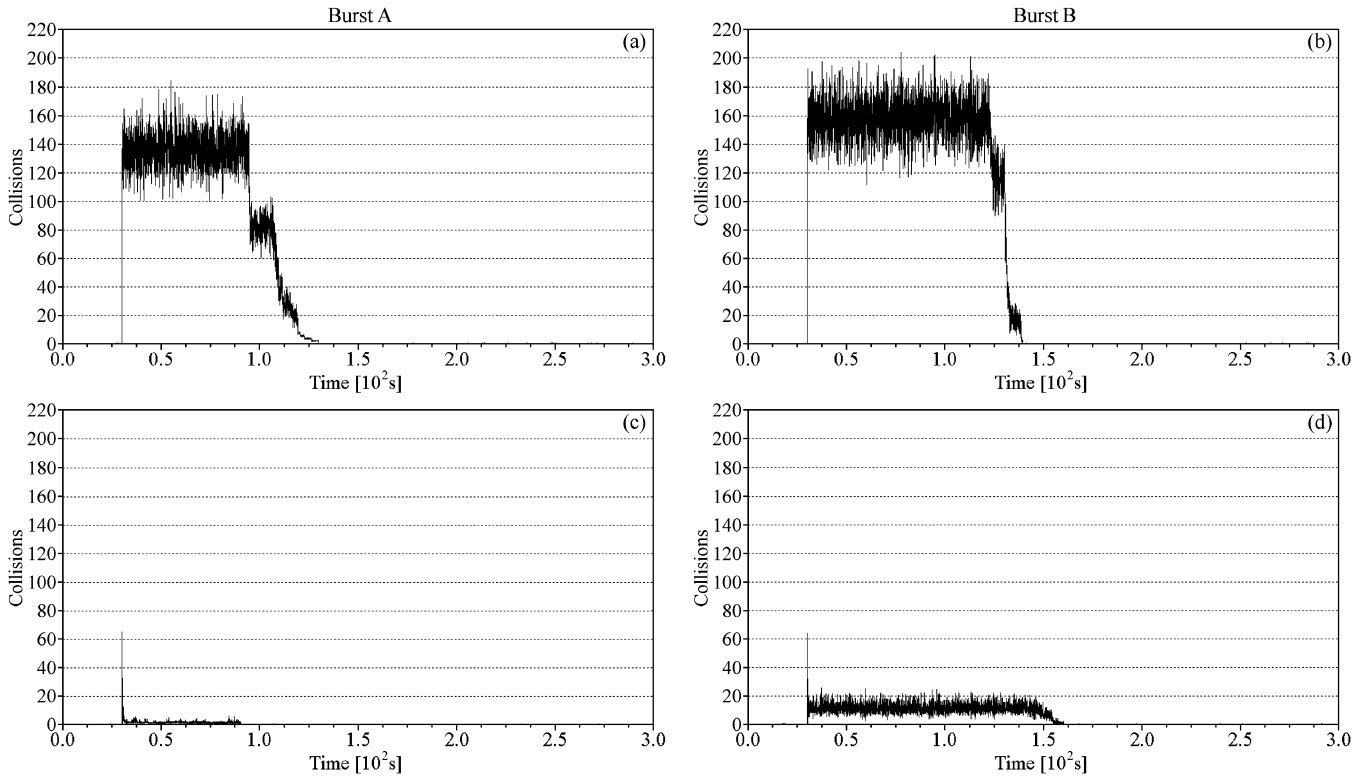

Fig. 4. Number of collisions with and without smoothing for Scenario 1. (a) Burst A, no smoothing. (b) Burst B, no smoothing. (c) Burst A, with smoothing. (d) Burst B, with smoothing.

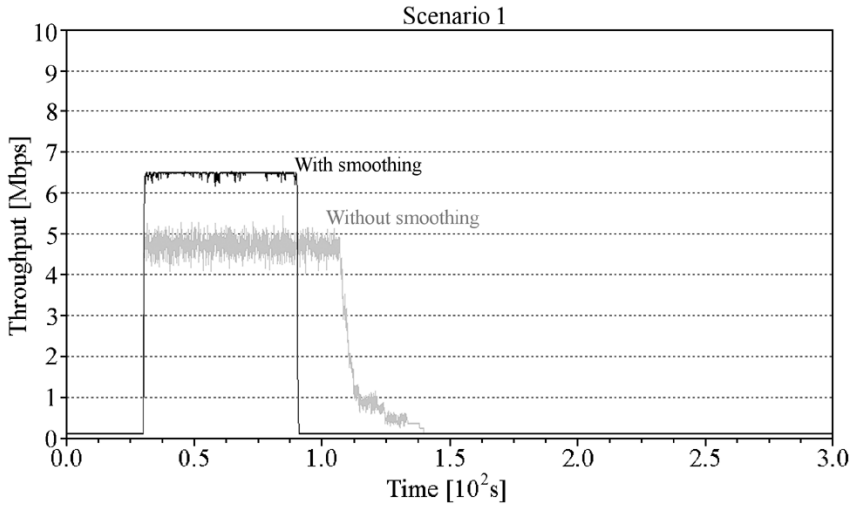

Fig. 5. Trace of throughput with Burst A with smoothing and without smoothing in Scenario 1.

smoother parameters, which achieved a good trade-off between the need to safeguard RT traffic delay and NRT throughput.

Scenario 2: In Scenario 2, the ten stations produce a total RT workload of 1.5 Mbps (the RT frame generation rate at each station being 50 frame/s, and the RT frame size at the Physical layer 3 Kbit). For NRT traffic, the workload is the same as Scenario 1. Scenario 2 therefore features more RT traffic, but with less urgent deadlines than the previous one (50 ms as compared with 1 ms). As much more RT traffic is generated than in the previous scenario, here, even when there are no bursts of NRT traffic, the number of collisions we measured was quite high and there were deadline misses, as shown in Table III (although the percentage was quite low: 4E-4). This is because no type of smoothing was applied to RT traffic, and under the heavy RT workload here considered, the probability of repeated collisions between RT packets is low, but not null. We then performed the same set of measurements as in Scenario 1, generating two bursts of NRT traffic: Burst A, which puts a heavy workload on the network (about 6.8 Mbps) and Burst B, which saturates the network by generating a workload of about 11 Mbps. For the sake of brevity, we will not show graphs, but discuss the results obtained, which are consistent with those shown in Scenario 1. Smoothing improves the overall throughput, included NRT one. Without smoothing, the

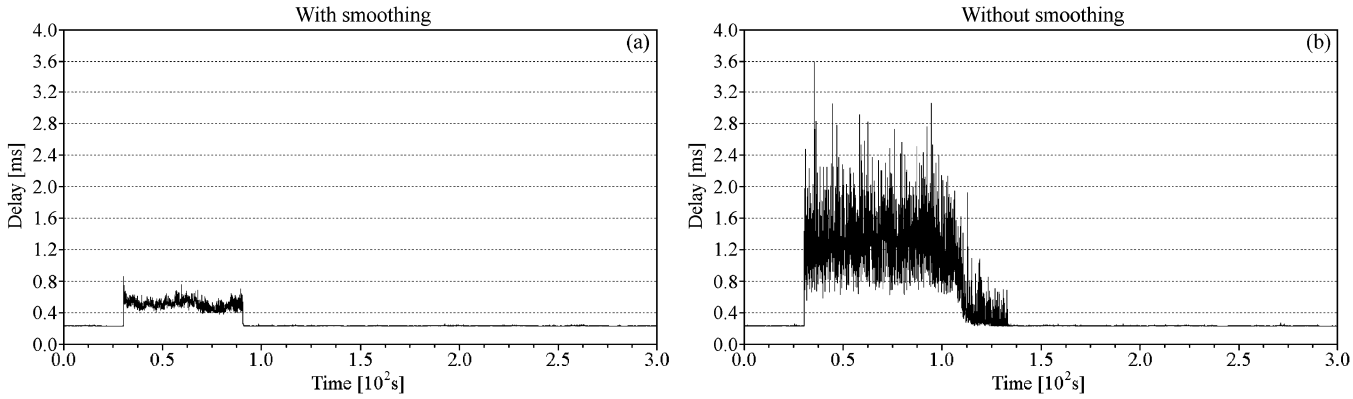

Fig. 6. Trace of RT traffic delivery delay for Burst A, Scenario 1. (a) With smoothing. (b) Without smoothing.

TABLE III
EXPERIMENTAL SCENARIOS AND DEADLINE MISS RATIO COMPARISON

RT scenario	NRT workload	Smoothing	D_{mr}
1	No	No	0%
1	Burst A	No	9.8%
1	Burst A	Yes	1.3%
1	Burst B	No	24%
1	Burst B	Yes	14%
2	No	No	4E-04%
2	Burst A	No	4.5%
2	Burst A	Yes	0.17%
2	Burst B	No	15%
2	Burst B	Yes	3.6%

NRT throughput burst is prolonged, because the traffic is occasionally blocked due to the high number of collisions and thus takes longer to serve. The peak throughput value (7.89 Mbps) is also higher with smoothing than without (about 6.1 Mbps). In the presence of smoothing, the RT delay grows to a limited extent and only during the workload burst. Without smoothing, the delay is much greater (more than two times) and prolonged beyond the workload burst, because the network takes much longer to deal with the burst owing to the heavy workload. The deadline miss ratio obtained with and without smoothing are summarized in Table III. These measures show once more the positive effect of smoothing on both RT and NRT traffic, with both kind of bursts.

V. CONCLUDING REMARKS

This paper addressed optimization of a fuzzy smoother by GAs, showing the effectiveness of the approach for a Shared Ethernet under different workload conditions. Nowadays, the trend is toward an extensive use of switches. However, it should be stressed that a switch by itself cannot guarantee deterministic RT behavior [18]–[20]. For example, in scenarios where the producer/consumer model is adopted (e.g., at the Field level, where such a model is quite common), switches handle producer/consumer interaction as broadcast traffic, and thus one of the major benefits deriving from the use of switches, that is, the existence of multiple simultaneous transmission paths, may be affected. In this context, [19] addresses the broadcast storms problem and the RT behavior of Ethernet switches in general. A set of practical experiments, carried out on an off-the-shelf Ethernet switch, showed some weaknesses affecting the RT behavior of Switched Ethernet, such as the low number of different priority levels provided by IEEE 802.1D [21] and IEEE 802.1Q [22] (up to eight distinct traffic classes to prioritize messages inside the switches), which is not enough to support efficient priority-based scheduling in general cases. The experiments also showed diverse sources of interference that RT traffic in a switch is subject to when the switch is heavily loaded. For example, it is possible for lower-prioritized traffic to lock the switch memory so that it cannot be used for higher-prioritized traffic. To solve the problem, the authors of [19] indicate traffic control, and in particular traffic smoothing, as a possible solution to be adopted inside switches. Given the promising nature of the topic, further work on fuzzy traffic smoothing will deal with fuzzy smoothers for Switched Ethernet networks.

APPENDIX
SEARCHING FOR THE PARETO FRONT TO VALIDATE THE OPTIMIZATION APPROACH

Given a set of N objectives to be maximized, a solution x is said to be *weakly dominated* by a solution y (where x is other than y) if the following relation holds:

$$f_j(x) \leq f_j(y) \quad \forall\, j = 1, \ldots, N \tag{A.1}$$

where x and y are the parameters to be optimized, whereas f_j is the objective function relating to the j-th objective. If x is weakly dominated by y, but there exists at least one index j^* such that

$$f_{j^*}(x) < f_{j^*}(y). \tag{A.2}$$

x is said to be *strongly dominated* by y. It is therefore preferable to y for part of the objectives to be maximized, and is no worse for the remaining objectives.

On the basis of the previous definitions, a solution is said to be a *Pareto* solution if it is not dominated (either weakly or strongly) by any other solution [23]. When a solution is a Pareto solution, it is not possible to obtain improvements in an objective without a consequent deterioration in performance for the other objectives. There is an infinite number of Pareto solutions: all those not dominated by others. This infinite number of solutions makes up what is called the *Pareto front* [16], [17],

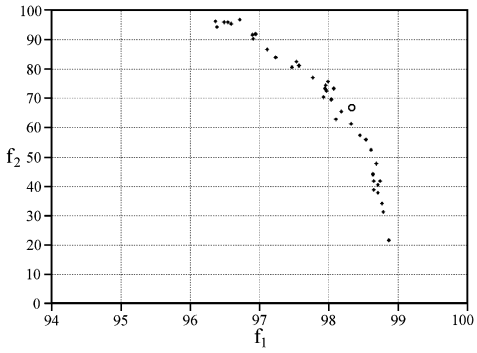

Fig. 7. Pareto front for Scenario 1 ($f_1 = 100 - D_{mr} \cdot 100$, $f_2 = E_b \cdot 100$).

which represents a bound, in our case an upper bound, to the possible solutions that can be found, which we will consider as being made up of strong solutions. The Pareto front shows the range of values to search for in order to find the solution which best matches the designer requirements. It is not necessary to find all the solutions to provide a qualitative definition of the front: it is sufficient to choose a set of an appropriate size. One of the best methods proposed to obtain the Pareto front using GAs is based on multi-objective optimization genetic algorithms (MOGAs), initially proposed in [24] and subsequently in [17], and then efficiently applied to the tuning of the parameters of a fuzzy controller in [25]. Here we implemented the MOGA technique to perform a search for the Pareto front of our optimization problem, thus enabling us to compare the results obtained and validate the choices made, especially the fitness function (3.5). Let us note that the Pareto front we are seeking is made up of solutions that are not *strongly dominated* by other solutions. Other techniques were also taken into consideration in our optimization, such as the use of vector evaluated genetic algorithms (VEGAs) [26], nondominated sorting genetic algorithms (NSGAs [27]), and other variations (for an extensive survey of the various techniques, the reader is referred to [16] and [17]). MOGAs base the search for and selection of solutions on the concept of *rank*: each member of the population is related to the others according to the principle of dominance, its rank corresponding to the number of individuals by which it is dominated. We used this technique as follows. Let us consider an individual x_i at generation t, which is dominated by $p_i(t)$ individuals in the current generation. Its current position in the rank of individuals can be given by

$$\text{rank}(x_i, t) = 1 + p_i(t). \tag{A.3}$$

All nondominated individuals are assigned a rank of 1, while dominated ones are penalized according to the population density of the corresponding region of the tradeoff surface. Fitness assignment is performed in the following way.

1) Sort population according to rank.
2) Assign fitness to individuals by interpolating from the best (rank 1) to the worst (rank $n^* < N$) according to some function, usually but not necessarily linear.
3) Average the fitnesses of individuals with the same rank, so that all of them will be sampled at the same rate.

This type of approach is highly selective and may cause premature convergence [17], [24]. We therefore modified the MOGA technique, adding *fitness sharing* [16], [17], which consists of penalizing solutions which yield results that are too close to each other in terms of fitness, to differentiate as much as possible between the number of optimal solutions found, thus covering the search space better. Fig. 7 shows the Pareto front we found for Scenario 1 (Section IV) using MOGAs. Scenario 1 was chosen as it is more critical than Scenario 2 for RT traffic (more urgent deadlines). The Pareto front provides the designer with a view of the search space from the perspective of the objectives to be optimized. We recall that here the objectives are the two components of the fitness function (A.4):

$$f(x) = 0.5 \cdot (100 - D_{mr} \cdot 100) + 0.5 \cdot (E_b \cdot 100) \tag{A.4}$$

more precisely:

$$f_1 = 100 - D_{mr} \cdot 100 \text{ and } f_2 = E_b \cdot 100. \tag{A.5}$$

Examining Fig. 7, we find that all the points belonging to the lefthand side of the front (which can be seen as a border line for the search space) are admissible solutions, but not optimal, while on the righthand side of the Pareto front no solutions exist. In Fig. 7, the solution we found using the fitness function (3.5) is depicted as a circle. The results obtained analyzing the Pareto front are consistent with those obtained using the Weighted Sum Approach [16], [17], although they are not exactly the same, as MOGAs tend to find a set of optimal solutions rather than a single solution. This proves that fitness function (3.5) is a good trade-off between the two values to be optimized.

REFERENCES

[1] L. Lo Bello and O. Mirabella, "Design issues for ethernet in automation," in *Proc. 8th IEEE Int. Conference on Emerging Technologies and Factory Automation, ETFA 2001*, Antibes Juan-les-pins, France, Oct. 2001, pp. 213–221.

[2] "802.3 CSMA/CD Access Method and Physical Layer Specification," IEEE, 1985.

[3] S. 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 Application Symp.*, Vancouver, BC, Canada, Jun. 1999.

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

[5] L. Lo Bello, M. Lorefice, O. Mirabella, and S. Oliveri, "Performance analysis of ethernet networks in the process control," in *Proc. 2000 IEEE Int. Symp. Industrial Electronics, ISIE 2000*, Puebla, Mexico, Dec. 2000.

[6] L. Lo Bello *et al.*, "Fuzzy traffic smoothing: An approach for real-time communication over ethernet networks," in *Proc. 4th IEEE Workshop on Factory Communication Systems, WFCS 2002*, Västeras, Sweden, Aug. 2002, pp. 241–248.

[7] M. Cho and K. G. Shin, "On soft real-time guarantees on the ethernet," in *Proc. 9th Int. Conf. Real-Time and Embedded Computing Systems and Applications, RTCSA 2003*, Tainan, Taiwan, R.O.C., Feb. 2003, pp. 59–76.

[8] The Network Driver Interface Specification (NIDS) Interface. [Online]. Available: http://www.microsoft.com

[9] J. S. Turner, "New directions in communications (or which way to the information Age?)," *IEEE Commun. Mag.*, vol. 24, no. 10, pp. 8–15, Oct. 1986.

[10] *Fuzzy Studio 2, Software Development Tool*, SGS-Thomson Microelectronics, 1995.

[11] R. Hill, B. Srinivasan, S. Pather, and D. Nihaus, "Temporal Resolution and Real-Time Extensions to Linux," Inform. Telecommun. Technol. Ctr., Univ. Kansas, Lawrence, Tech. Rep., kurt-utime-1998, ITTC-FY98-TR-11 510-03, 1998.

[12] U. Bodenhofer and F. Herrera, "Ten lectures on genetic fuzzy systems," in *Preprints of the International Summer School: Advanced Control-Fuzzy, Neural, Genetic*, R. Mesiar, Ed. Bratislava, Slovakia: Slovak Technical Univ., 1997, pp. 1–69.
[13] R. Alcalá, J. Casillas, O. Cordón, F. Herrera, and J. S. I. Zwir, "Techniques for learning and tuning fuzzy rule-based systems for linguistic modeling and their application knowledge engineering," in *Systems, Techniques and Applications*, C. T. Leondes, Ed. New York: Academic, 1999.
[14] T. L. Seng, M. Khalid, and R. Yusof, "Tuning of a neuro-fuzzy controller by genetic algorithm," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 29, no. 2, pp. 226–236, Apr. 1999.
[15] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.
[16] C. C. Coello, "Using the min-max method to solve multiobjective optimization problems with genetic algorithms," in *IBERAMIA'98*, Lecture Notes in Computer Science, Lisbon, Portugal, 1998, pp. 303–314.
[17] C. M. M. da Fonseca, "Multiobjective Genetic Algorithms with Application to Control Engineering Problems," Ph.D. thesis, Dept. Automat. Contr. Syst. Eng., Univ. Sheffield, U.K., Sep. 1995.
[18] J. D. Decotignie, "A perspective on ethernet-TCP/IP as a fieldbus," in *Proc. of the 4th FeT 2001 International Conference on Fieldbus Systems and their Applications*, Nancy, France, Nov. 2001, pp. 138–143.
[19] P. Pedreiras, R. Leite, and L. Almeida, "Characterizing the real-time behavior of prioritized switched-ethernet," in *Proc. RTLIA03, the 2nd Int. Workshop on Real-Time LAN's in the Internet Age, Satellite Workshop of the 15th IEEE Euromicro Conf. Real-Time Systems*, Porto, Portugal, Jun. 2003, pp. 59–66.
[20] J. Loeser and H. Haertig, "Low-latency hard real-time communication over switched ethernet," in *Proc. 16th IEEE Euromicro Conf. Real-Time Systems, ECRTS 2004*, Catania, Italy, Jun. 30– Jul. 2, 2004, pp. 13–22.
[21] *Information Technology—Telecommunications and Information Exchange Between Systems-Local and Metropolitan Area Networks-Common Specification-Media Access Control (MAC) Bridges*, (ISO/IEC) ANSI/IEEE Std 802.1D.
[22] *1998 IEEE Standard for Local and Metropolitan Area Networks*, IEEE 802.1Q.
[23] I. S. Sbalzarini, S. Muller, and P. Koumoutsakos, "Multiobjective optimization using evolutionary algorithms," in *Proc. 2000 Summer Program*. Stanford, CA, Nov. 2000.
[24] C. M. M. da Fonseca and P. J. Fleming, "An overview of evolutionary algorithms in multiobjective optimization," *Evol. Comput.*, vol. 3, pp. 1–16, 1995.
[25] T. Sayers, J. Anderson, and S. Clement, "The multi-objective optimization of a traffic control system," in *Proc. 14th Int. Symp. Transportation and Traffic Theory*, A. Ceder, Ed., Haifa, Israel, Jul. 1999, pp. 153–176.
[26] J. D. Schaffer, "Multiple objective optimization with vector evaluated genetic algorithms," in *Proc. First Int. Conf. Genetic Algorithms and Their Applications*, Hillsdale, NJ, 1985, pp. 93–100.
[27] N. Srinivas and K. Deb, "Multiobjective optimization using nondominated sorting in genetic algorithms," *Evol. Comput.*, vol. 2, no. 3, pp. 221–248, 1994.

Lucia Lo Bello (M'02) received the M.S. degree in electronic engineering in 1994 and the Ph.D. degree in computer engineering in 1998, both from the University of Catania, Italy.

During the academic year 2000–2001, she was a Visiting Researcher, Department of Computer Engineering, Seoul National University, Seoul, Korea. She is currently an Assistant Professor with tenure in the Department of Computer Engineering and Telecommunications, University of Catania. Her research interests include real-time systems, factory communications, distributed process control systems and embedded systems. Since 1994, she has authored more than 45 technical papers published in the proceedings of international conferences, books and journals in the area of real-time systems, factory communications and distributed systems.

Prof. Lo Bello is a member of Subcommittee 65C, Working Group 11, of the International Electrotechnical Commission (IEC), working on the standardization of Real-Time Ethernet (RTE). She is a member of the IEEE Industrial Electronics Society and of the Technical Committee on Factory Automation, where she serves as Co-Chair of the Subcommittee on Intelligent Sensors and Sensor Networks in Industrial & Factory Automation.

Giordano Antonio Kaczyński received the M.S. degree in computer engineering in 2004 from the University of Catania, Italy. He is currently a Ph.D. student at the Computer Engineering Department, University of Catania.

Orazio Mirabella received the M.S. degree in physics from the University of Catania, Italy.

He is currently a Full Professor of computer networks with the Engineering Faculty, University of Catania. His research interests include real-time distributed systems, communication protocols, Fieldbuses, ad-hoc wireless networks and sensor networks. Since 1982 he has been active in IEC TC65C/WG6 contributing to the definition of a Standard Fieldbus. He is the author of more than 120 technical papers published in international journals and conferences proceedings.