Fuzzy Traffic Smoothing: Another Step towards Statistical Real-Time Communication over Ethernet Networks

R. Caponetto[+], L. Lo Bello*, O. Mirabella*

*Dipartimento di Ingegneria Informatica e delle Telecomunicazioni
+Dipartimento di Ingegneria Elettrica Elettronica e Sistemistica
Università di Catania, ITALY
riccardo.caponetto@dees.unict.it, {llobello,omirabel}@diit.unict.it

Abstract

The paper presents an improvement on existing dynamic traffic smoothing techniques in two respects. Firstly, here the input parameters for the smoother are both the overall throughput and the number of collisions observed over an interval. Together, these two parameters represent a more complete indicator of the actual network workload. Secondly, here the smoothing action is dynamically gauged according to the actual workload by using a *fuzzy controller*.

Experimental results in a real environment, comprising 11 workstations running the Linux O.S. and connected via a 10BASE-T Ethernet, are presented, together with a performance comparison with a dynamic smoother in the literature.

1. Introduction

The main obstacle to using Ethernet in real-time communication is that, due to the CSMA/CD access protocol, Ethernet cannot provide connected stations with deterministic channel access times and therefore guarantee that data delivery deadlines will be met. As Ethernet technology today offers a number of appealing features, which suggest adopting it even in time-constrained environments, recently a lot of research addressed the problem of enforcing a predictable behaviour on Ethernet networks.

To support soft real-time applications, which do not require determinism and can accept a *statistical* bound on packet delivery time Shared Ethernet can be adopted, provided that a suitable mechanism to statistically guarantee deadline meeting to real-time packets is implemented. Here we deal with a mechanism, called *traffic smoothing*, which was introduced in [1] and is based on the definition of statistical real-time channels running on an Ethernet. As is known, in an Ethernet, the delay a packet undergoes depends on the number of attempts to transmit before transmission is successfully achieved. In [1] it is demonstrated that a sufficient condition to guarantee with a pre-fixed probability that a packet will gain access to the channel by a pre-established time is that the total rate of new packets generated by the stations remain below a threshold called the *network-wide input limit*. As the Ethernet MAC protocol is totally distributed, a single station is not aware of the current packet arrival rate for the whole network.

Thus, in order to maintain the *network-wide input limit,* each station is assigned a *station input limit* calculated according to the packets' deadlines and the tolerable packet-loss ratio. Each station regulates the *packet stream* arriving from the Application layer in such a way as to keep the packet arrival rate at its MAC sub-layer below the *station input limit* it has been assigned. *Traffic smoothing* only acts on non-real-time (NRT) packets to smooth traffic bursts, as when packets arrive in bursts they are more likely to collide. Traffic smoothing is realised locally in each Ethernet station by a software layer called a *traffic smoother*, inserted between the TCP/IP and the Data Link layer (Fig.1), which buffers any NRT packets arriving in a burst and sends them in such a way to keep their arrival rate at the MAC layer below the *station input limit*.

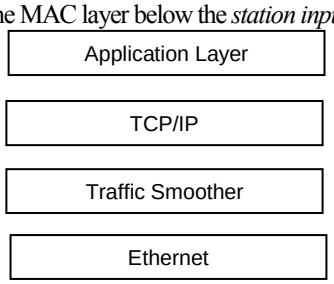

Fig. 1. Software architecture of traffic smoothing

The traffic smoother is implemented by using a leaky bucket-based algorithm [6], where a *credit bucket depth* (CBD), which indicates the capacity of the credit bucket, and a *refresh period* (RP) are defined. Every RP seconds, up to CBD credits are replenished. The CBD/RP ratio is the *station input limit* and determines the average throughput available for a station. By varying the values of RP or CBD, it is possible to control the bursty nature of a flow of packets and thus of the traffic generated by the single stations. When a NRT packet arrives from the IP layer, if there is at least one credit in the bucket, the traffic smoother sends it to the Ethernet Network Interface Card (NIC) and removes a number of credits equal to the size of the packet in bytes. Otherwise, the packet is not transmitted until the next replenishment. A real-time (RT) packet is not affected by smoothing, but its transmission does consume credits. This means that if there is both RT and NRT traffic in a station, the latter is transmitted using any credits that are left over after the transmission of the RT traffic for that station. In *static* traffic

smoothing [1], the *station input limit* is assigned to each station once and for all in such a way that, even in the worst case, each station can be provided with a statistical guarantee on the timely delivery of its packets. This solution has the drawback of entailing a considerable waste of bandwidth, as the *station input limit* is assigned to each station irrespective of the actual load currently on the network (which can even be significantly below the *network-wide input limit*, as not all stations are necessarily transmitting at any one time). Also, scalability problems may arise when the number of stations is high. To provide more scalability and better bandwidth exploitation, in *dynamic* traffic smoothing the [2][3] the *station input limit* is dynamically adapted according to the network workload, thus the available bandwidth is shared only among the stations which really need to transmit. In order to evaluate the network load for dynamic smoothing purposes, different approaches have been proposed in the literature [2][3]. In [2] a dynamic smoother based on throughput control is dealt with, which adapts the refresh period RP to the network throughput measured over a given time interval, while keeping the CBD value fixed. On the other hand, the dynamic smoother described in [3] applies the *harmonic-increase and multiplicative-decrease* (HIMD) algorithm to react to the detection of a *single* collision over an interval. According to the HIMD adaptation, when a packet collision is detected, the RP is increased by the minimum between twice its current value and a given RP_{max} value, while in the absence of collisions the RP is periodically decreased (with period τ) by a constant Δ down to a value of RP_{min}.

Here we propose an approach which extends the previous work outlined above in two respects. Firstly, we use both the total throughput and the number of collisions as input parameters for the smoother. Together, these two parameters represent a more complete indicator of the actual workload. For example, one criticism that can be made of the approach proposed in [3], which reacts to the detection of a single collision over an interval of length α, is that the occurrence of a single collision is not necessarily due to network congestion, but may derive from a temporal coincidence between the transmission requirements of two or more stations when the load on the network is not particularly high. For this reason, regulation of the traffic generation rate should also take total throughput into account. According to the throughput value, in fact, doubling the RP may be too drastic or excessively penalising for NRT traffic.

Another significant improvement introduced here, as compared with the approaches in [2] and [3], lies in the fact that RP regulation is not statically fixed, but is dynamically varied and gauged according to the actual workload by using a *fuzzy controller*. The motivation for choosing a fuzzy controller is that the system considered here, due to its non-linear and quite complex behaviour, is difficult to control using traditional controllers, but is highly suitable for control that is capable of integrating the heuristic knowledge acquired in the field by experts. The choice of a fuzzy approach allows us to embed the knowledge of an expert in the controller. Thus, unlike the approaches in [2][3], where RP regulation is based on fixed variations (doubling the RP or decreasing it by a constant Δ), the action here varies and is differentiated according to the values simultaneously taken by the *two* inputs.

2. Fuzzy smoothing

Fig. 2 gives a detailed block scheme of the fuzzy controller.

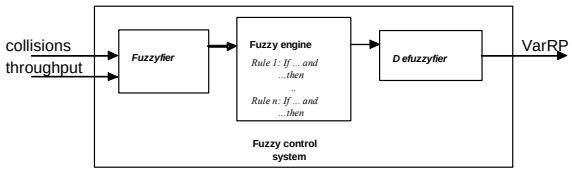

Fig. 2. The fuzzy controller

Our fuzzy controller has two inputs, i.e. the number of collisions and the overall throughput observed in a reference interval, and a single output, i.e. the quantity by which to vary the *refresh period* according to the input values, here called *VarRP*. If RP_{old} is the current *refresh period*, the new value RP_{new} is:

$$RP_{new} = RP_{old} + VarRP \quad (1)$$

Three *membership functions* were defined for each input, corresponding respectively to the values (*Low, Med, High*). This number was chosen heuristically as a tradeoff between representing all the possible operating modes of the considered system in relation to the values taken by the inputs and avoiding 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. Thus, the output variable was assigned nine different membership functions, one for each rule. The membership functions here chosen for the input variables were triangular and trapezoidal, as they are the most suitable for representing the type of inputs being examined. The structure of the inference rules, indicating the control action to apply according to all the possible combinations of the input variables, is:

If Collisions IS ... AND Throughput IS ... THEN VarRP IS ...

For the output variable, here *singleton* membership functions were chosen, i.e. each set comprises a single point with a degree of membership of 1 (*crisp*). It has to be noticed that the fuzzy controller output is not a single value among the nine defined in the rule set, as the fuzzy controller interpolates, according to fuzzy arithmetic, all the nine crisp output variables. The fuzzy controller was tuned using heuristic *trial and error* procedures.

3. Experimental evaluation

The behaviour of the *fuzzy* smoothing was investigated in a real scenario comprising 11 workstations equipped with the Linux O.S. and connected via a 10BASE-T Ethernet. The collision domain diameter is 10-metre. We implemented both the fuzzy smoother and the HIMD one, and performed a performance comparison in the same environment and operating conditions outlined in [3]. In both cases, the

smoothing driver was activated on each node with an RP_{max} of 100 ms, an 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 [3], here we measured the *roundtrip delay* for the RT control messages exchanged between two processes, called Client and Server, running on two different PC_S, 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 the Station processes. In [7] a complete description of the experimental scenario is given. Here, for the sake of brevity, we just show the results for comparative purposes. Figs. 3 and 4 show how the roundtrip delay and the throughput 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 confirmed by the throughput curve (Fig. 4), which is lower but broader than the workload one. 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.

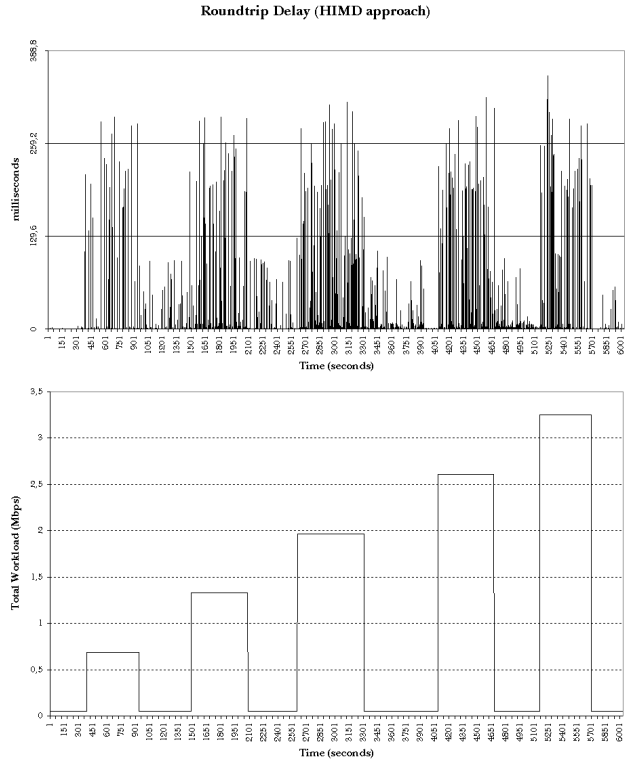

Fig. 3. Roundtrip delay for RT messages (HIMD)

Fig.3 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.

The *fuzzy* controller used in this experiment comprised the *inference rules* shown in Table 1. Fig. 5 gives the roundtrip values for the single messages with varying workloads from all the processes, both RT and NRT, obtained using the fuzzy smoother. Comparison with Fig. 3 shows that here the delays are much shorter, even when there are bursts. Very few messages feature roundtrip delay values over 129.6 ms. 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. 6, which shows a more regular trend of total throughput vs. the workload and higher throughput values than the graph in Fig. 4. 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, we compared deadline miss ratio (Dmr) obtained with the two approaches. With a *deadline* of 129,6 ms, the Dmr using the fuzzy smoother never exceeds 0,7%, about 1/3 of the one obtained using HIMD. This result is highly satisfactory for many soft RT applications.

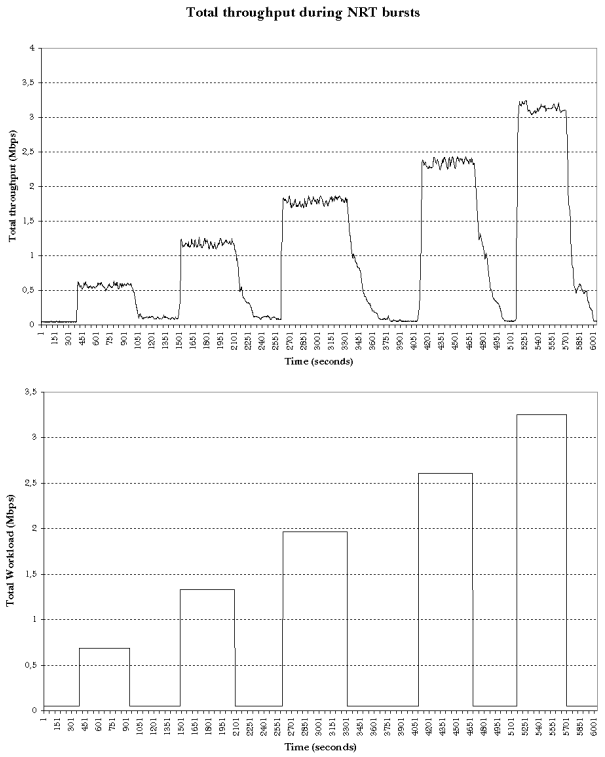

Fig.4. Total network throughput (HIMD)

Table. 1. The inference rules used

ANTECENDENT	CONSEQUENT
If Collisions IS Low AND Throughput IS Low THEN *VarRP* IS	$-RP_{min}$ *0.5
If Collisions IS Low AND Throughput IS Med THEN *VarRP* IS	$-RP_{min}$*0.3
If Collisions IS Low AND Throughput IS High THEN *VarRP* IS	0
If Collisions IS Med AND Throughput IS Low THEN *VarRP* IS	$-RP_{min}$*0.1
If Collisions IS Med AND Throughput IS Med THEN *VarRP* IS	0
If Collisions IS Med AND Throughput IS High THEN *VarRP* IS	$+RP_{max}$*0.2
If Collisions IS High AND Throughput IS Low THEN *VarRP* IS	$+RP_{max}$
If Collisions IS High AND Throughput IS Med THEN *VarRP* IS	$+RP_{max}$*0.7
If Collisions IS High AND Throughput IS High THEN *VarRP* IS	$+RP_{max}$*0.6

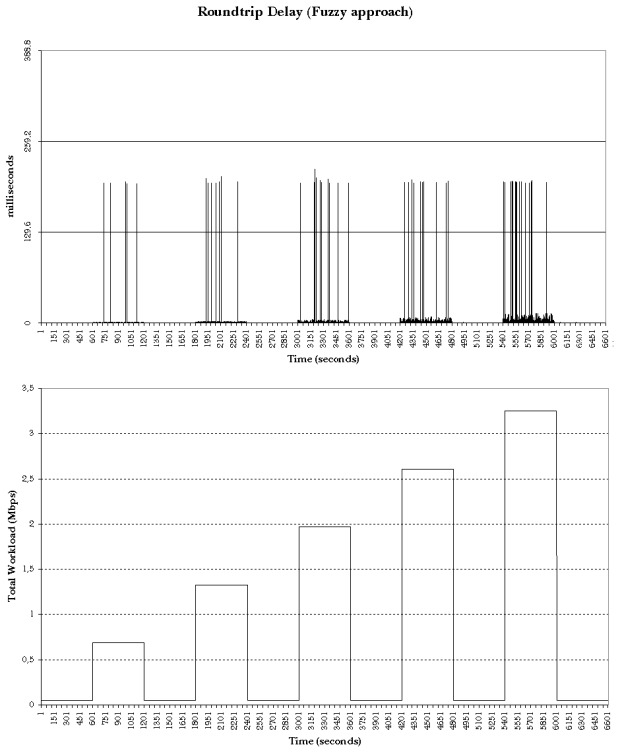

Fig. 5. Roundtrip delay for RT messages (fuzzy)

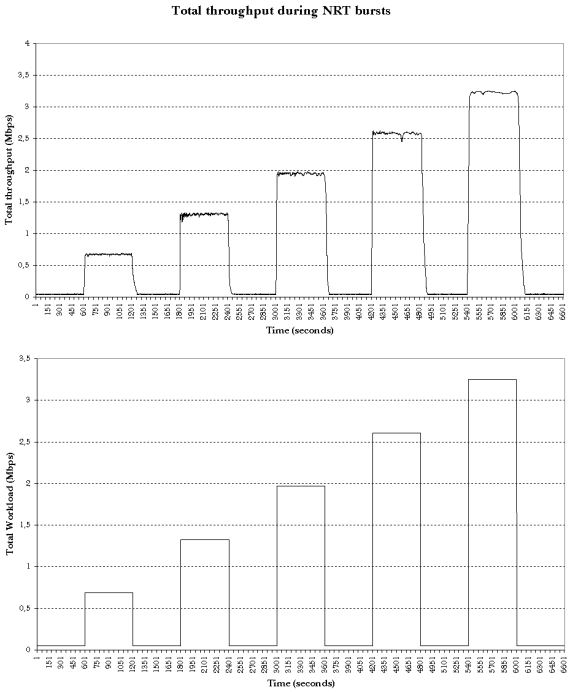

Fig. 6. Total network throughput (fuzzy)

4. Conclusions

Experimental results obtained in a real environment have confirmed the benefits of the approach proposed.

References

[1]S. Kweon, K.G. Shin, Q. Zheng, "Statistical Real-Time Communication over Ethernet for Manufacturing Automation Systems", *Proc. of RTAS'99*, June 1999, Vancouver, Canada.

[2]L. Lo Bello, M. Lorefice, O. Mirabella, S. Oliveri, "Performance Analysis of Ethernet Networks in the Process Control", *Proc. of IECON'00,* Puebla, Mexico, Dec. 2000.

[3] S. Kweon, K. G. Shin, et al. "Achieving Real-Time Communication over Ethernet with Adaptive Traffic Smoothing", in *Proc. of RTAS 2000*, pp.90-100, Washington DC, USA, June 2000.

[4]Yager R., Zadeh L.A. Editors, *An Introduction to Fuzzy Logic Applications in Intelligent systems*, Kluwer Academic, 1992.

[6] R.L. Cruz, "A Calculus for network delay, Part I: Network Elements in Isolation", *IEEE Trans. on Information Theory*, 37(1), Jan 1991.

[7] A. Carpenzano, R. Caponetto, L. Lo Bello, O. Mirabella, "Fuzzy Traffic Smoothing: an Approach for Real-time Communication over Ethernet Networks", to appear on *Proc. of WFCS'02,* Västerås, Sweden, August 2002.