A Closed-loop Fuzzy Traffic Controller for Fair Bandwidth Sharing

Dario Pompili* **and Francesco Delli Priscoli†**
*Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey, Piscataway, NJ 08854
†Department of System Engineering
University of Rome "La Sapienza," 00184 Rome, Italy
e-mail: *pompili@ece.rutgers.edu, †dellipriscoli@dis.uniroma1.it

Abstract— **This paper introduces an innovative closed-loop traffic controller to efficiently and fairly share a single common resource, e.g., the available bit rate in a wireless link, among a set of traffic flows characterized by different Quality of Service (QoS) requirements. The proposed traffic controller relies on a control-based approach that exploits a closed-loop feedback architecture including Proportional-Integral-Derivative (PID) controllers that are dynamically tuned using fuzzy logic. The traffic controller is able to react to different scenarios and traffic load conditions, while targeting the twofold objective of maximizing the resource utilization and respecting the flow QoS requirements. Simulation results demonstrate that the proposed closed-loop traffic controller outperforms a well-known open-loop traffic controller (based on Dual Leaky Buckets (DLBs) and Earliest Deadline First (EDF) scheduling algorithm).**

Index Terms— **Control Theory, PID Controller, IP Traffic, Fuzzy Logic, Quality of Service.**

I. Introduction

ONE of the main challenges of the out coming telecommunication networks is the provision of Quality of Service (QoS) guarantees to IP flows, which, most of times, are still served according to a best-effort paradigm that does not assure any QoS guarantee. The respect of pre-defined QoS guarantees, already a challenging goal in wired networks, is even more challenging in wireless networks where this goal has to be achieved in conjunction with a high utilization of the valuable available bandwidth. As a result, in wireless networks, traffic control strategies play a key role to assure QoS guarantees and, at the same time, to efficiently exploit the available bandwidth.

This paper proposes an innovative closed-loop traffic controller that assures a high utilization of the wireless available bandwidth while allowing the respect of the traffic QoS guarantees. The proposed controller has been developed and tested for a transparent satellite platform, i.e., no routing functionalities are performed on the satellite. Nevertheless, the proposed solution can be profitably used in new generation IP-based satellite platforms [1] and, more in general, in bandwidth-limited wireless networks where fairness among admitted traffic flows and the respect of their QoS requirements are paramount. Figure 1 shows the satellite reference scenario considered in this paper, where various traffic flows coming from the IP backbone network reach an Hub Station (HS), which is in charge of forwarding them towards the satellite through the wireless channel. The goal is to *efficiently* and *fairly* share a single common resource, i.e., the uplink bandwidth, which is pre-assigned to the considered HS, among the various incoming IP flows, while meeting their QoS requirements.

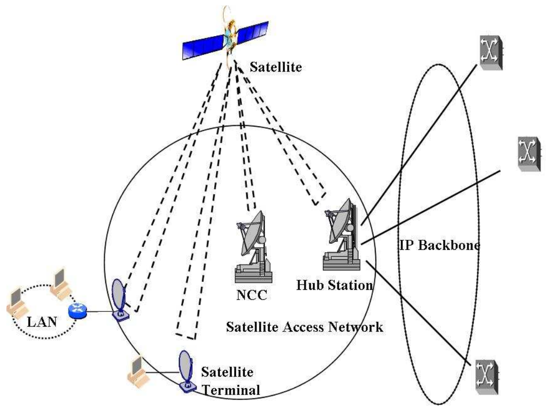

Fig. 1. Satellite reference scenario

The proposed approach relies on the presence of a control-based closed-loop traffic controller, which continuously computes a parameter providing an updated measurement of the overall efficiency experienced at the HS, as well as of the efficiencies experienced by each IP flow, i.e., the satisfaction of the flow need for bandwidth. Then, appropriate closed-loop control actions steer the single IP flow efficiencies to track the overall efficiency experienced at the HS. This approach assures both *flow fairness*[1] and *high bandwidth utilization*, as will be shown in the paper. The application of a well-established control strategy using Proportional-Integral-Derivative (PID) controllers in conjunction with an innovative fuzzy tuning approach, rather than adopting complex and sometimes cumbersome optimization techniques such as adaptive learning, neural networks, and nonlinear control, is shown through extensive simulation experiments to lead to very satisfying results for the network management problem investigated in this paper. It is worth noting that, unlike simpler controllers, PID controllers give accurate and stable control by adjusting process outputs based

[1]*Flow fairness* measures how equally different traffic flows, which are characterized by the same QoS requirements, share a common network resource.

on the history and rate of change of the error signal, i.e., the difference between system and flow efficiencies. Moreover, in contrast to more complex algorithms such as optimal control theory, PID controllers can be adjusted on-line, without advanced mathematics. To the best of our knowledge, this is the first paper to apply a classical closed-loop control technique enhanced by fuzzy logic to efficiently and fairly share the available bandwidth in a wireless channel.

The remainder of the paper is organized as follows. Section II presents the traffic controller architecture and details its main components. Section III focuses on the closed-loop control algorithm, while Section IV shows the performance results. Finally, Section V drives the main conclusions.

II. Closed-loop Traffic Controller Architecture

The goal of the proposed traffic controller is to *efficiently* and *fairly* share the available uplink bandwidth so as to serve all the accepted IP flows while respecting their QoS requirements. As far as the QoS requirements are concerned, the most widely used approach in the literature is to identify a set of *QoS classes* [2] and map the IP flows to these classes according to their specific QoS requirements. Specifically, each IP flow is associated with a QoS class by setting a proper field in the header of all its packets. Note that IP flows with slightly different QoS requirements can be mapped into the same class. A QoS class is characterized by a set of QoS guarantees [3][4] (hereafter, also refereed to as *QoS contract*), namely:

1) A minimum bit rate, R_{min}, called *Compliant Bit Rate*, which must be granted to the flows belonging to the QoS class in question, whatever the traffic condition might be;
2) A maximum and a minimum transfer delays, indicated as D_{max} and D_{min}, respectively, which packets belonging to the flows in question must experience.

Traffic exceeding the *Compliant Bit Rate* is referred to as *Non Compliant*. According to the policy strategy, the system may serve this traffic as best effort, without having to comply with its QoS requirements. Note that the constraint on the maximum tolerated transfer delay comes from the requirement of guaranteeing IP packet delivery within given delays, while the constraint on the minimum transfer delay comes from the requirement of limiting the delay jitter J (whose maximal value is indicated hereafter as J_{max}=D_{max}-D_{min}).

The internal structure of the Hub Station (HS) traffic controller is shown in Fig. 2. The IP packets coming from the Internet, i.e., the *offered traffic*, are sorted into a set of FIFO (First in First out) queues. A one-to-one association between queues and QoS classes exists, i.e., each IP packet is enqueued into the FIFO queue associated with the class the packet belongs to. The core of the traffic controller is the closed-loop controller, namely *"Main Controller"*, which takes the decisions concerning the packets to be retrieved from the FIFO queues for being forwarded towards the air interface, i.e., towards the satellite. The closed-loop controller takes as inputs the following parameters, which are monitored every $T_{control}$ seconds (referred to as *control period*):

1) *Queue Lengths* of the FIFO queues;
2) *Loss Bit Rates*, i.e., the bit rates of the traffic that is discarded by the FIFO queues (either because packets expired or because the queues are full);

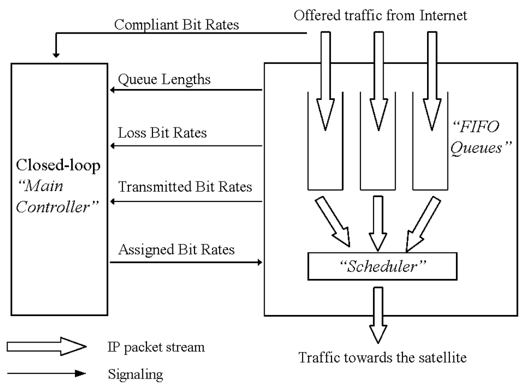

Fig. 2. Internal architecture of the HS controller, including the *"Main Controller"*, *"Scheduler"*, and *"FIFO Queues"*

3) *Transmitted Bit Rates*, i.e., the bit rates of the traffic that is retrieved from the FIFO queues and transmitted through the physical interface.

Periodically, every *control period*, the main controller, basing on these three parameters and taking the *Compliant Bit Rate* into account, computes the *Assigned Bit Rates*, i.e., the bit rates that can be granted to the various flows in the next control period, as shown in Fig. 2. The *Assigned Bit Rates* are expressed as fractions of the *Total HS Bit Rate*, i.e., the bit rate that is currently assigned to the considered HS. The *Assigned Bit Rates* are communicated by the *"Main Controller"* to the *"Scheduler"*, which is in charge of retrieving the packets from the FIFO queues and forwarding them towards the satellite. By doing this, the scheduler tries to comply with the bit rate assignments decided by the main controller.

Note that the main controller assumes a *continuous-flow model* of the inbound traffic, i.e., it bases its computations on continuous measures of bit rates and queue bit lengths. This means that it overlooks the *discrete nature* of IP packets. Conversely, the scheduler takes the non-continuous nature of IP packets into account. This causes a small difference between the bit rates actually assigned to the IP flows by the scheduler and the bit rates decided by the main controller. In this respect, the scheduler implements a proper *bit credit system* that aims at keeping the error between these two bit rates close to zero. Note that packets that have waited in the FIFO queues more than their maximum tolerated transfer delay D_{max}, i.e., that are expired, are discarded to save precious bandwidth and prevent traffic congestion. The amount of discarded traffic over a control period is the so-called *Loss Bit Rate*.

III. Main Controller Algorithm

As stated in the previous section, the control action performed by the main controller takes place at the beginning of every control period, i.e., every $T_{control}$ seconds. Conceptually, the control action consists of two stages:

1) *Compliant Bit Rate* Assignment,
2) *Non Compliant Bit Rate* Assignment.

In the first stage (detailed in Section III-A) the main controller assigns a fraction of the *Total HS Bit Rate* ($TotalHSR$) to the IP flows according to their *Compliant Bit Rates*, which are defined by their QoS Contract. Then, in the second stage

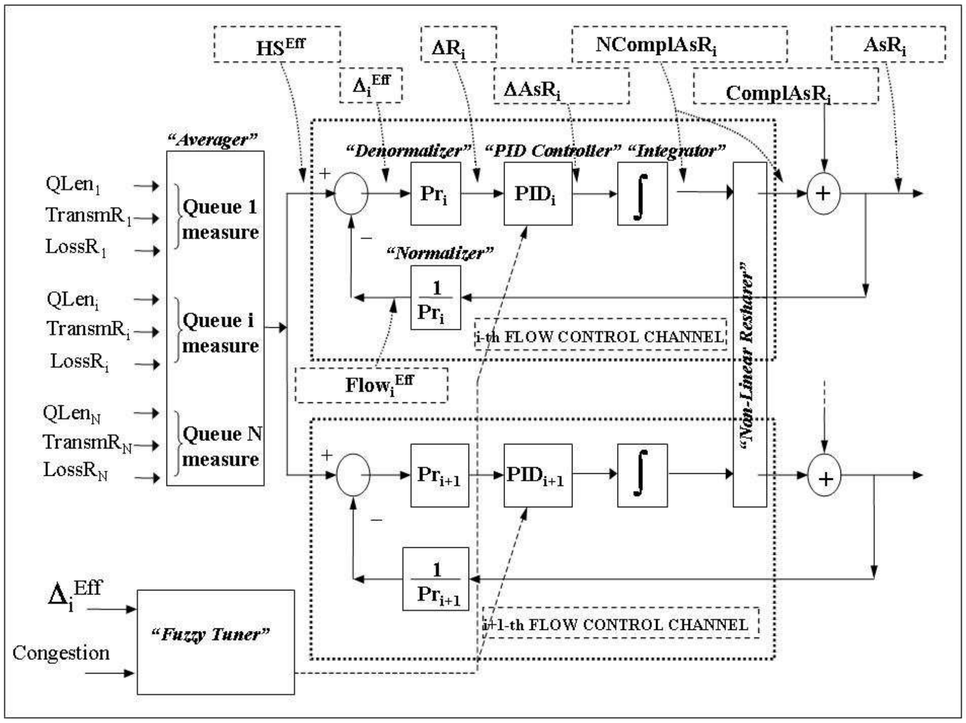

Fig. 3. Closed-loop HS controller and its components: *"Averager"*, *"Denormalizer"*, *"PID Controller"*, *"Integrator"*, *"Non-linear Resharer"*, *"Normalizer"*, and *"Fuzzy Tuner"*

(detailed in Section III-B), the main controller assigns the remaining fraction of $TotalHSR$, according to the algorithm described in the following. The sum of the two above-mentioned bit rates, which are assigned to the i^{th} flow, will be hereafter referred to as $AssignedRate_i$ (AsR_i). This value will be used throughout the following control period, until a new AsR_i is computed. In the following, the different variables of interest will be referred to the i^{th} IP flow and to the control period and, for the sake of clarity, will be indicated with self-explaining shortened symbols.

A. First Stage: Compliant Bit Rate Assignment

In the first stage of the control action the controller checks the $OfferedRate_i$ of the i^{th} flow (i.e., the average bit rate offered by the IP flow in the last control period) and compares it with its *Compliant Bit Rate*, granting to the i^{th} flow a $CompliantAssignedRate_i$ ($ComplAsR_i$) equal to the lower of the two values (so as to comply with the QoS contract, while avoiding waste of bandwidth). The rationale behind this choice is to assign output bit rate on the base of the input (offered) bit rate, and exploit the gain from statistically multiplexing different uncorrelated flows. Then, the main controller computes the $SpareHSRate$ ($SpareHSR$), which is equal to $TotalHSR$ minus the sum of all $ComplAsR_i$, i.e.,

$$SpareHSR = TotalHSR - \sum_i ComplAsR_i. \qquad (1)$$

Note that this value is always positive since the sum of the *Compliant Bit Rates* is always less than the $TotalHSRate$, as guaranteed by a proper Connection Admission Control (CAC) mechanism. While the development of efficient CAC strategies is out if the scope of this paper, the reader is referred to [5] for further details on this topic.

In the second stage, $SpareHSR$ has to be partitioned into the $NonCompliantAssignedRate_i$ ($NComplAsR_i$) according to the algorithm detailed in the following section. Clearly, for a generic IP flow i, the total $AssignedRate_i$ (AsR_i) is equal to

$$AsR_i = ComplAsR_i + NComplAsR_i. \qquad (2)$$

B. Second Stage: Non Compliant Bit Rate Assignment

This second stage is the core of the control action and the main novelty of this paper. The approach used by the main controller to assign the $SpareHSRate$ to the IP flows is based on comparing the status of each flow against the overall HS status, with the objective of increasing or decreasing the bandwidth assignment to each flow to assure fairness among all flows and comply with their QoS requirements. Figure 3 shows the overall control scheme of this second stage, under the assumption that the considered HS is simultaneously handling N IP flows. For the sake of clarity, the description of the control algorithm is split into the following four steps.

1) Computation of Priority and Efficiency of a Single Flow: The first step of the algorithm is the definition of a flow metric, indicated as $Priority$ (Pr_i), which takes the current *"need for bandwidth"* of the i^{th} flow into account, as

$$Pr_i = \frac{QLen_i}{T_{control}} + TransmR_i + LossR_i. \qquad (3)$$

Note that if both sides of (3) were multiplied by $T_{control}$, the left side of the equation ($Pr_i \cdot T_{control}$) would be equal to the sum of the current number of bits in the queue associated with the i^{th} flow, $QueueLength_i$ ($QLen_i$), and the number of bits that have been transmitted ($TransmR_i \cdot T_{control}$) and discarded ($LossR_i \cdot T_{control}$) in the *last control period*. In other words, $Pr_i \cdot T_{control}$ represents the total number of bits that would be stored in an *ideal queue* if all the arrived bits were enqueued (under the assumption that no transmissions nor discards occur in the entire length of the control period).

Now, we introduce the metric $FlowEfficiency_i$ ($Flow_i^{Eff}$), computed by the component named *"Normalizer"* in Fig. 3, which normalizes the bit rate assigned to a flow taking its *Priority*, i.e.. its need for bandwidth, into account, as

$$Flow_i^{Eff} = \frac{AsR_i}{Pr_i}. \tag{4}$$

When $Flow_i^{Eff} \approx 1$ in (4), it means that the bit rate assigned to the i^{th} IP flow is consistent with its *actual* need for bandwidth, expressed by Pr_i in (3). Conversely, when $Flow_i^{Eff} \gg 1$ (or $Flow_i^{Eff} \ll 1$) the i^{th} flow has more (or less) bandwidth, i.e., assigned bit rates, than it needs to satisfy its QoS requirements.

2) Computation of HS Priority and HS Efficiency: In the second step of the algorithm, the definitions of the metrics $Priority_i$ and $Flow_i^{Eff}$ are extended to the entire HS. The objective is to obtain average metrics of the overall system that can be compared against the metrics associated with the single flows. To do this, we introduce two system metrics, $HSPriority$ ($HSPr$) and $HSEfficiency$ (HS^{Eff}), where

$$HSPr = \sum_{i=1}^{N} Pr_i, \tag{5}$$

$$HS^{Eff} = \frac{TotalHSR}{HSPr}. \tag{6}$$

Note that (6) is similar to (4), where the single flow characteristics have been replaced with the characteristics of the overall system. All the computations relevant to the first and the second steps are performed by the *"Averager"* component on the left side in Fig. 3.

3) Closed-loop Computation of $AssignedRate_i$ (AsR_i): As we said, the basic idea behind our closed-loop controller is to compute the efficiency of each IP flow and compare it against the HS overall efficiency: the difference between these two efficiencies is used to achieve *fairness* among flows by assessing whether to increase or decrease the bandwidth assigned to each flow. Referring to the i^{th} flow, the algorithm computes, in a closed-loop fashion, the AsR_i (i.e., the total bit rate that the i^{th} flow will be assigned in the next control period) using the following five-step procedure (see Fig. 3):

- HS^{Eff} and $Flow_i^{Eff}$ are compared to assess if the i^{th} flow is *under-served* or *over-served*. The difference between these two values is the so-called $\Delta Efficiency_i$ ($\Delta_i^{Eff} = HS^{Eff} - Flow_i^{Eff}$): positive values of Δ_i^{Eff} mean that the i^{th} flow is under-served, and that its efficiency must be increased, and viceversa;
- Δ_i^{Eff} is multiplied by Pr_i to obtain $\Delta Rate_i$ (ΔR_i), which represents the bit rate assignment variation that will align the i^{th} flow efficiency with the overall HS efficiency;
- ΔR_i is fed to a *"PID Controller"*, whose parameters are dynamically controlled by a *"Fuzzy Tuner"* (see Fig. 4), as detailed in section III-B.4. The result is $\Delta AssignedRate_i$ (ΔAsR_i);
- ΔAsR_i is summed to $NComplAsR_i$ computed in the previous control period;
- After a precautionary non-linear resharing aiming at avoiding negative values of $NComplAsR_i$, which could decrease $ComplAsR_i$ and cause the infringement of the QoS contract relevant to the i^{th} flow, $NCompAsR_i$ is summed to $ComplAsR_i$, thus obtaining AsR_i, as in (2).

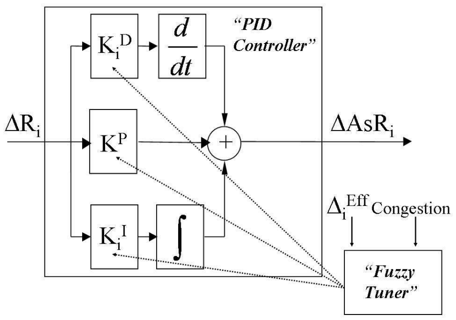

Fig. 4. *"PID Controller"* relevant to the i^{th} IP flow and *"Fuzzy Tuner"*

4) PID Control and Fuzzy Tuning: Fig. 4 shows the PID controller relevant to the i^{th} flow, whose parameters (K^P, K_i^I, and K_i^D) are controlled by the *"Fuzzy Tuner"*, which implements three different sets of rules relevant to the *Proportional*, *Integral*, and *Derivative Actions*, as detailed in the following.

Proportional Action: The first set of rules tunes the aggressiveness of PID controllers, which is regulated by the K^P parameter. To handle the immediate error, i.e., $\Delta_i^{Eff} \cdot Pr_i = \Delta R_i$, the error is multiplied by the constant K^P. The various flows strive to obtain a fraction of the shared bandwidth: when there are many flows with respect to the available bandwidth, many of them may be under-served; this would cause high requests of bandwidth from most of the current active flows, which ultimately may result in the risk of oscillation and instability. The system manages to reach the stability by reducing the aggressiveness of PID controllers.

In order to evaluate the system traffic load, and therefore assessing the aggressiveness needed from PID controllers, we introduce the *Congestion* metric, which depends on two variables, *Need* and *Surplus*, as

$$Congestion = \frac{Need}{Need + Surplus}, \tag{7}$$

where

$$\begin{aligned} Need &= \textstyle\sum_{i=1}^{N} \max[\Delta AsR_i; 0], \\ Surplus &= \textstyle\sum_{i=1}^{N} \max[-\Delta AsR_i; 0]. \end{aligned} \tag{8}$$

Note that, in (8), $Need$ and $Surplus$ represent the cumulative request for bandwidth of all the flows that are *under-served* and *over-served*, respectively.

A problem faced with PID controllers is that they are linear, and that their performance in non-linear systems, such as the bandwidth assignment dealt in this paper, is variable. For this reason, we decided to enhance PID controllers by using *fuzzy logic*[2]. Specifically, we exploit the fuzzy logic to dynamically

[2] *"What is striking is that its most important and visible application today is in a realm not anticipated when fuzzy logic was conceived, namely, the realm of fuzzy-logic-based process control"* [6].

control the PID parameters, i.e., K^P, K_i^I, and K_i^D. The basic idea of the Fuzzy Logic Control (FLC) was suggested in [7]. FLC provides a non-analytic alternative to the classical analytic control theory. By utilizing simple triangular fuzzy sets [8] and the Sugeno deduction [9], we implemented three rules to decrease K^P as *Congestion* increases, and to decrease K_i^I and K_i^D as Δ_i^{Eff} decreases. These last two rules prevent newly established IP flows from pulling pre-existing flows away from the equilibrium. As new flows have null assigned bit rate, in fact, their priority is very high at the moment of inception and may tend to overcome bandwidth requests form existing flows. Note that when the error is zero, a proportional controller's output is zero. This means that the proportional action cannot always guarantee that the setpoint, i.e., HS^{Eff}, will be reached if the setpoint is not fixed in time. This is called a *Steady State Error*. To fix this, integral and derivative actions are required. The tuning of the PID integral and derivative parameters, i.e., K_i^I and K_i^D, depends on the status of the i^{th} flow, while the proportional parameter K^D is regulated according to the overall system status.

Integral Action: The second set of fuzzy rules controls the K_i^I parameter, according to the principles of PID tuning [10]. To learn from the past, the error, i.e., ΔR_i, is integrated and multiplied by the constant K_i^I. High values of K_i^I busts the integral action, while too high values can cause instability. Without integral term, a PID controller cannot eliminate the error if the process requires a non-null input to produce the desired setpoint. Specifically, the integral action is used to speed the control up when the system is distant from equilibrium, and to turn it off when the equilibrium is approached. In particular, this action is useful in the transients when the FIFO queues fill up: the approximately linear increase of the queue length relevant to the i^{th} flow, caused by a constant bit rate[3], triggers the almost-linear increase of $Priority_i$, according to (3), which is properly controlled by the *"Integrator"* component.

Derivative Action: The third set of fuzzy rules controls the K_i^D parameter according to [11]. To anticipate the future, the first derivative (the slope of the error, i.e., $\frac{d\Delta R_i}{dt}$) over time is calculated and multiplied by the constant K_i^D. In general, by increasing K_i^D, the loop more quickly reach its reference after a load disturbance. However, too much derivative action will cause excessive response and overshoot. In our system, the derivative action is shown to introduce minor benefits to the overall performance.

IV. Performance Evaluation

The overall control scheme has been simulated using OPNET [12], a discrete-event network simulator. Since non-linear fuzzy controllers make infeasible a mathematical analysis of the system performance, we tested and refined our solution through extensive simulation experiments. All the results are averaged over many runs to assure small relative confidence intervals. In the performance evaluation we considered four QoS classes: Voice, FTP, Video, and Web-browsing. The statistical parameters characterizing IP flows belonging to the four considered classes, as well as their average and peak bit rates, are reported in Table I, while Table II reports the parameters characterizing the QoS contract of the considered traffic classes. As far as the simulation scenario is concerned, we considered the satellite system depicted in Fig. 1.

TABLE I
IP Traffic Source Statistical Parameters*

	L_P[Byte]	T_{int}[ms]	R_{av}[Kbps]	R_{max}[Kbps]
Voice	72	20	29	29
FTP	$\mathcal{U}[40, 4960]$	$\mathcal{U}[50, 150]$	200	400
Web	$\mathcal{U}[40, 2960]$	$\mathcal{U}[50, 550]$	40	240
Video	$\mathcal{U}[40, 2960]$	$\mathcal{N}(8.1, 20)$	1350	1460

*L_P: packet length, T_{int}: packet inter-arrival time, R_{av} and R_{max}: average and peak rates, $\mathcal{N}$ and $\mathcal{U}$: gaussian and uniform distributions

TABLE II
IP Flow QoS Contract

	D_{max}[s]	D_{min}[s]	J_{max}[s]	R_{min}[Kbps]
Voice	0.10	0.01	0.09	29
FTP	4.00	0.20	3.80	150
Web	1.50	0.05	1.45	30
Video	0.50	0.02	0.48	1000

The performance achieved by the proposed closed-loop controller is compared with the one of a *FIFO controller* and of an *open-loop controller*. In the *FIFO controller* only one queue is considered: all IP packets are enqueued in the same queue and are served according to a First In First Out (FIFO) discipline, without taking the flow QoS requirements into account. This simple, but quite common and effective, controller allows studying the disadvantages in scenarios where no QoS classes are considered. Conversely, in the *open-loop controller* [13] the incoming IP traffic is sorted, according to its QoS class, to a set of Dual Leaky Buckets (DLBs) [3] that are used to identify the compliant traffic, which is then served, on a *first priority basis*, according to an Earliest Deadline First (EDF) algorithm [14]. The non compliant traffic is served, on a *second priority basis*, again according to an EDF algorithm. The name of the controller is due to the fact that all the DLB parameters are statically set according to closed-form formulas exploiting the concept of *equivalent bandwidth*, without any closed-loop mechanism. While open-loop scheduling algorithms can perform well in static or dynamic systems in which the workloads can be accurately modeled [15], in our experiments they are shown to perform poorly. This is due to the unpredictability of the considered dynamic systems, mainly caused by the burstyness of traffic sources.

Figure 5 graphs the *Link Utilization* (i.e., the ratio of the used and available link bandwidth) achieved by the competing controllers as a function of the *System Congestion*, which is defined as the ratio of the $TotalHSRate$ and the average bit rates of the overall offered traffic. As can be seen in Fig. 5, the proposed closed-loop controller outperforms the other controllers (in every possible load condition), and manages to achieve near ideal performance. This is proven by the fact that the *Link Utilization* of the closed-loop algorithm closely follows

[3]Note that bit rates from bursty sources (FTP, Web-browsing, Compressed Video, etc.) can be considered *almost constant* during a burst, while bit rates from stable sources (Voice, Uncompressed Video) have a little variance around their mean value and can be considered, again, almost constant.

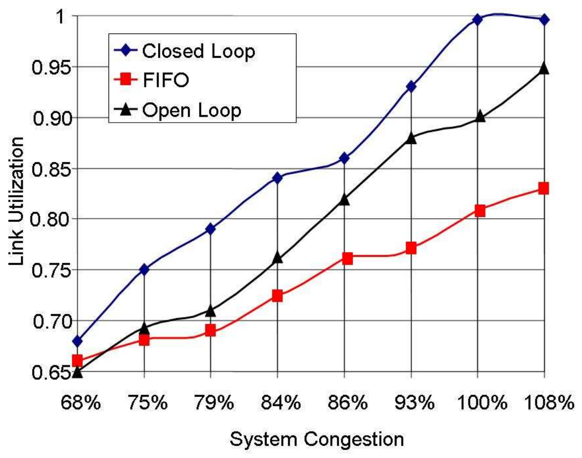

Fig. 5. Link Utilization

TABLE III

EVOLUTION OF TRAFFIC CLASS EFFICIENCIES VS. SYSTEM EFFICIENCY$^+$

Sim. Time [s]	3	6	9	12	15	18
System Efficiency	**0.50**	**0.25**	**0.15**	**0.10**	**0.07**	**0.06**
Voice Efficiency	0.60	0.32	0.18	0.10	0.08	0.07
FTP Efficiency	0.40	0.20	0.09	0.04	0.05	0.06
Web Efficiency	0.70	0.30	0.12	0.05	0.06	0.07
Video Efficiency	0.30	0.18	0.12	0.09	0.07	0.06

$^+$System Congestion=108%

the *System Congestion* for every traffic load less than 108%, and that it is always above the link utilizations of its competing schemes.

Table III presents the evolution of the average efficiency of Voice, FTP, Web, and Video traffic classes vs. the reference system efficiency. Note the difference between the smooth behavior of Voice/Video connections, and the burstlike evolution of FTP/Video connections. Although the system is affected by high congestion (System Congestion equal to 108%), i.e., there is less available bandwidth than that needed to accommodate all traffic data, the proposed closed-loop controller manages to smoothly bring the connection efficiencies back to the reference value, thus providing fairness and minimizing the loss bit rates.

Finally, Table IV reports the average delays and delay jitters of successfully received IP packets according to the three competing controllers when the System Congestion is 93%, i.e., when the system capacity is almost fully exploited but the system is not yet in a congested state. As far as the closed-loop controller is concerned, the QoS constraints are never infringed and the IP packet loss is close to zero. The results confirm that our scheme exploits better the available bandwidth since it can dynamically adapt its parameters, i.e. its working point, according to the traffic statistical characteristics.

TABLE IV

AVERAGE DELAY AND DELAY JITTER*

Delay $\div$ Jitter [s]	Closed Loop	FIFO	Open Loop
Voice	$0.05 \div 0.04$	$0.09 \div 0.04$	$0.07 \div 0.07$
FTP	$2.10 \div 1.50$	$3.20 \div 2.30$	$2.70 \div 1.80$
Web	$1.10 \div 0.80$	$1.70 \div 0.70$	$1.30 \div 1.10$
Video	$0.30 \div 0.20$	$0.40 \div 0.30$	$0.30 \div 0.20$

*System Congestion=93%

To summarize, the closed-loop controller presented in this paper manages to outperform the other considered bandwidth sharing algorithms because of three factors:

1) The definition of a proper band-sharing mathematical schematization that highlights the drivers of need for bandwidth in wireless connections;
2) The utilization of a closed-loop controller that results in a high degree of adaptivity to the dynamic system and in some degree of traffic forecasting;
3) The implementation of a smart scheduler that properly realizes the calculated band-sharing on the IP packets.

V. CONCLUSIONS

We presented an innovative closed-loop traffic controller to share the available bit rate among a set of traffic flows characterized by different QoS requirements. The proposed controller relies on a control-based approach that exploits a closed-loop feedback architecture including PID controllers tuned that are dynamically tuned using fuzzy logic. We showed that our control scheme outperforms competing open-loop schemes, and that it is able to minimize the loss bit rate and maximize the efficiency of the wireless channel under different scenarios and traffic load conditions.

REFERENCES

[1] T. Inzerilli and D. Pompili, “Towards a New Generation of IP-based Satellites,” in *Proc. of EU Information Society Technology Mobile and Wireless Communications Summit (IST Summit)*, Copenhagen, Denmark, Nov. 2002.
[2] N. K. Blake, S. Baker, and D. Black, “Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers,” IETF RFC 2474, Tech. Rep., Dec. 1998.
[3] R. Braden, D. Clark, and S. Shenker, “Integrated Services in the Internet Architecture: an Overview,” IETF RFC 1633, Tech. Rep., Jul. 1994.
[4] “TIPHON: Definition of QoS Classes,” ETSI TS 101 329-2, Tech. Rep., Jul. 2000.
[5] F. Delli Priscoli and A. Isidori, “A Control-Engineering Approach to Traffic Control in Wireless Networks,” in *Proc. of Conference on Decision and Control (DCD)*, Las Vegas, Dec. 2002.
[6] L. A. Zadeh, “Fuzzy Logic,” *IEEE Computer Mag.*, vol. SMC-3, no. 1, pp. 83–93, Apr. 1988.
[7] ——, “Outline of a New Approach to the Analysis of Complex Systems and Decision Processes,” *IEEE Trans. Syst., Man., Cyber.*, vol. SMC-3, no. 1, pp. 28–44, 1973.
[8] R. Q. Hu and D. W. Petr, “A predictive self-tuning fuzzy-logic feedback rate controller,” *IEEE/ACM Trans. Netw.*, vol. 8, no. 6, pp. 697–709, 2000.
[9] M. Sugeno, *An Introductory Survey of Fuzzy Control.* Information Science, 1985.
[10] K. J. Astrom and T. Hagglund, *PID Controllers: Theory, Design and Tuning*. North Caroline: Instrument Society of America, 1995.
[11] V. B. Baji’c, “Simple Rule-based and Fuzzy Controllers,” TR95-07, Control Laboratory, Technikon Natal, RSA, Tech. Rep., 1995.
[12] OPNET, http://www.opnet.com/.
[13] J. Rexford, F. Bonomi, A. G. Greenberg, and A. Wong, “A Scalable Architecture for Fair Leaky Bucket Shaping,” in *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, 1997, pp. 1054–1062.
[14] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son, “Design and Evaluation of a Feedback Control EDF Scheduling Algorithm,” in *Proc. of the IEEE Real-Time Systems Symposium*. IEEE Computer Society, 1999, pp. 56–67.
[15] Z. J. Haas, “Design Methodologies for Adaptive and Multimedia Networks,” *IEEE Communications Magazine*, vol. 39, no. 11, pp. 106–107, Nov. 2001.