AN API FOR ADVANCED TRAFFIC CONTROL IN DIFFSERV ROUTERS*

Giuseppe Bianchi, Vincenzo Mancuso
University of Palermo (Italy)
bianchi@elet.polimi.it; vincenzo.mancuso@tti.unipa.it

Paolo Di Francesco
CRES - Monreale, Palermo (Italy)
difrancesco@cres.it

Abstract Distributed per-flow admission control is a promising solution for Differentiated Services networks. Its deployment in DiffServ domains requires the ability to suitably configure, in each network router, low-level packet forwarding mechanisms, such as packet dropping algorithms driven by traffic measurements. In this paper we first show that performance effectiveness is achieved by means of non traditional configuration of the forwarding mechanisms. Hence, we propose a modular Application Program Interface that allows to flexibly and adaptively configure the forwarding/dropping behavior associated to a router's output queue, well beyond the traditional RED/RIO active queue management schemes.

1. Introduction

Market competition requires the co-existence of various ISP types. These range from premium ISPs, aimed at providing high quality and tight traffic control in change of costly peering agreements, down to free-of-charge ISPs aimed at maximizing their link utilization. In such a competitive scenario, the Differentiated Services (DiffServ) framework [1] provides new directions in the deployment of Quality of Service scenarios. In DiffServ, QoS is provided by the combination of several low-level mechanisms such as packet schedulers, droppers, meters. Different

*This research has been supported by the European Community and MIUR, Italy, in the frame of the POLLENS project (ITEA project if00011a - Platform for Open, Light, Legible & Efficient Network Services).

administrative domains should be made capable of dynamically tuning their DiffServ routers, i.e. they should have available a large set of router configuration parameters to achieve the desired QoS provisioning level.

This paper focuses on the packet control (dropping/forwarding) function of a DiffServ router, i.e. the set of low-level packet forwarding mechanisms devised to exert packet level traffic control, and manage congestion situations in the router. Traditional packet dropping mechanisms, such as RED (Random Early Discard) or RIO (Random Early Discard with IN/OUT packets) have been proposed to improve congestion control in the presence of TCP flows [2]. The idea is that congestion can be more effectively controlled by dropping a few packets at routers (i.e. by making a few TCP sources react) before the queue becomes full. In doing this, TCP mechanisms running at the network border will seldomly encounter situations of packet loss bursts, which are proved to be dramatic for the TCP performance [3, 4].

Within the DiffServ framework, packet dropping mechanisms find a practical application in the Assured Forwarding Per-Hop Behavior (AF-PHB) specification [5]. Despite the generality of the AF-PHB specification, it is frequently assumed that the AF-PHB will be implemented at each core router as a RIO mechanism. As a consequence, most current implementations [6, 7] are based on a monolithic approach. The queue is seen as a single entity, where measurement mechanisms (filtering and smoothing of the queue occupation status) are tightly integrated with packet dropping functions, and both are indivisible entities from the pure buffering capabilities of the queue. The only way to marginally operate on such an implementation is via the specific RIO configuration thresholds and parameters.

However, the apparently consolidated scenario described above does not account for novel ways to use low-level mechanisms. Specifically, it has been recently shown [8] that the same forwarding/dropping mechanisms built in the AF-PHB specification allow the support of stateless and distributed admission control within a DiffServ domain (see section 2). This in turns implies that the capability of carefully tuning the configuration of a mere packet forwarding mechanism translates into the much more important capability of setting the QoS provisioning level for an administrative domain (i.e. by rejecting incoming traffic when the network reaches a predetermined congestion level).

Unfortunately, when traditional RED/RIO mechanisms are used for distributed admission control support, the resulting performance is poor [9]. Hence, it emerges the need for more flexible and modular node-level Application Program Interfaces (APIs), that allow the deployment of non traditional low-level forwarding/dropping mechanisms, specifically devised for admission control purposes. Clearly, the same APIs would find application in the deployment of improved Active Queue Management schemes (e.g. adaptive RED, etc).

The rest of the paper is organized as follows. Section 2 briefly reviews how a distributed admission control function can be built upon low-level packet forwarding mechanisms implemented at the DiffServ core routers. Section 3 introduces novel forwarding/dropping mechanisms, and shows their performance effectiveness. Section 4 presents a flexible and modular architecture and related API that allows to support the considered schemes, as well as traditional mechanisms. Finally, conclusions and further research issues are given in section 5.

2. Packet dropping mechanisms for distributed admission control support

In the last few years, there has been quite a large amount of literature on distributed admission control in DiffServ [10, 11, 12]. An approach which allows the deployment of an effective - and tunable - distributed admission control scheme over a fully conforming DiffServ network has been presented in [8] under the name GRIP (Gauge&Gate Reservation with Independent Probing).

The basic idea of GRIP is elementary. When a new flow is offered to a domain, the end-point sends in the network a suitably labelled "probing" packet. The flow is admitted only if the probing packet is received at the destination node and relayed back to the sender. Similarly to TCP, failed reception of probing packets is interpreted as congestion in the network. GRIP becomes effective as long as two conditions jointly occur. First, each DiffServ router along the path must distinguish a probe from an information packet. This is achieved by marking probe packets with a different DiffServ label than data packet. Second, the performance of GRIP is related to the capability of routers to locally take decisions about the degree of congestion, and suitably block probing packets when congestion conditions are detected. To this purpose, the congestion level can be computed via local traffic measurements on the accepted traffic.

What makes GRIP compatible with DiffServ is the use of probe dropping/forwarding (i.e. implicit signalling, rather than explicit signalling), to convey a reject/accept information at the sender node. Specifically, GRIP can be deployed over the AF-PHB specification as follows. We recall that AF-PHB specifies four aggregate traffic classes. Within each class, three drop levels are considered. Let AFxi be the drop level i of AF class x. The relation among the drop levels is such that, within a class x, if $i < j$, the dropping probability of packets labeled AFxi is lower than that of packets labeled AFxj. Let us now assume that one AF class, say class x, is reserved for admission controlled traffic (indeed note that more than one class may be used to manage different flow aggregates, with different target QoS settings). To support GRIP, it is sufficient to i) mark information packets as AF$x1$; ii) mark probe packets as AF$x2$; iii) set the drop level for AF$x2$ packets on the basis of run-time measurements taken on the AF$x1$ aggregate flow [8].

The described operation translates the problem of provisioning a given QoS level within a domain, into the much more simple problem of suitably configuring a packet forwarding/dropping mechanism within each domain's router. Specifically, the definition of a suitable dropping rule is fundamental in terms of provisioned QoS. For example, a domain can configure its routers so that all AFx2 packets are dropped (i.e. all incoming calls are blocked) when the average AFx1 traffic exceeds a given threshold. The higher the threshold, the worse the QoS performance provisioned.

In the following section, we evaluate the performance of a more general RED-like dropping profile enforced on the AF$x2$ packet class. Specifically, when the load is below a min threshold, all the AF$x2$ packets are forwarded, while they are all dropped if the load is above a max threshold. Between these two thresholds, a linear dropping probability is enforced, i.e., being $B \in (min, max)$ the measured load, the dropping probability on the AF$x2$ packets is set to $(B - min)/(max - min)$. B is conveniently obtained as a smoothed version of the instantaneous load. In particular, in the numerical results, we have adopted a discrete time scale, with sample time $T = 100$ ms, and, at each time step k, we have evaluated $B(k)$ as:

$$B(k) = \alpha B(k-1) + (1-\alpha)M(k)$$

where $M(k)$ is the traffic load, in bits/sec, attempting to enter the link buffer during the time slot k. We chose $\alpha = 0.99$, corresponding to about 10 s time constant in the filter memory.

As illustrated in the following section, the additional AF$x3$ drop level available in the AF-PHB specification may be optionally used (with an associated additional dropping profile) instead of AF$x2$ when a more congestion-sensitive probing is needed, i.e. when a connection requires a greater amount of resources.

3. Performance Evaluation

The simulation results presented in this section consider a single 2 Mbps bottleneck link. Two different traffic scenarios have been considered. In the first scenario, flows are homogeneous. Each traffic source is modelled as VBR source, alternating heavy-tailed ON/OFF periods. While in the ON state, a source transmits at a Peak Constant Rate (PCR) randomly generated in the small interval 31 to 33 Kbps. Conversely, while in the OFF state, it remains idle.

The same ON/OFF profile has been considered in the second scenario, but the PCR of the sources has been uniformly generated in the range 20 to 80 Kbps. In this second scenario, we have additionally considered the possibility to adopt, as PROBE label, the AFx3 packet marking for flows whose PCR was greater than 40 Kbps. Conformingly, we have set a more restrictive drop profile for the AFx3 drop level. The rationale behind this choice is that a flow with higher PCR is more critical, in terms of resource consumption, than a flow with small PCR. However, the described GRIP operation is not able to distinguish, during flow set-up, flows with different PCRs (signalling is implicit - no traffic specification parameters are notified to the router). By using an additional AFx drop level, we are able to differentiate flows during their probing phase. In particular, the router has been configured so that a (high PCR) flow that probes the network via an AFx3 packet will be blocked in advance with respect to a low PCR flow.

Since we used an infinite buffer size QoS is characterized by the delay experienced by data packets rather than packet loss as in [13]. The throughput/delay performance of the adopted admission control scheme is evaluated in figures 1 to 5, for various packet dropping algorithm configurations. 99-th delay percentiles have been considered as delay performance metric.

Figure 1 shows the sensibility of the admission control scheme as the dropping profile changes. The *max* threshold is fixed to 95%, while the *min* ranges from 50% to 95%. For each threshold configuration, different points are obtained by tuning the offered load from 20% to 90%. Better delay performance are achieved when the minimum and maximum thresholds are distant from each other. As a reference comparison, the figure reports also results obtained by assuming that a standard - parameter based[1] - admission control mechanism is adopted. The fact that parameter-based admission control performs worse than a mechanism based on run-time measures, when traffic is long range dependent, has been thoroughly discussed and motivated in [14].

[1]In a parameter-based admission control scheme, the admission control decision is computed on the basis of the link status (in the homogeneous case, number of already admitted calls) as well as the known - or signalled -traffic source characteristics (peak, average, burst size, etc)

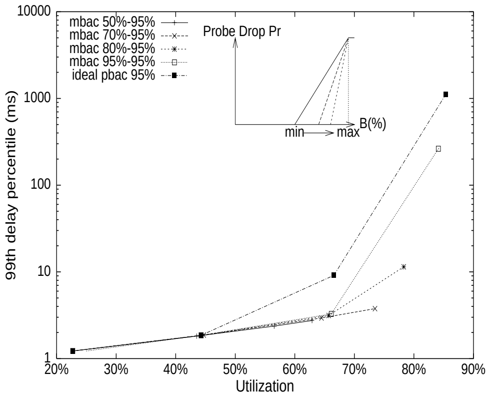

Figure 1. Offered load: 20% to 90%; source PCR: ~ 32 Kbps

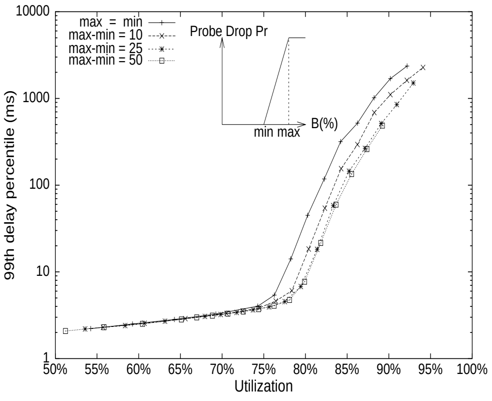

Figure 2. Offered load: 650%; source PCR: ~ 32 Kbps

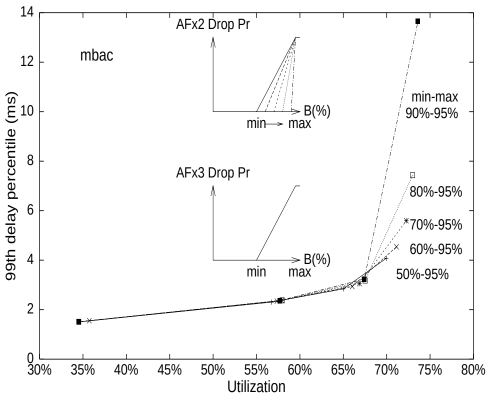

Figure 3. Offered load: 30% to 130%; source PCR: 20 to 80 Kbps; sources with PCR greater than 40 Kbps use AFx3 probes

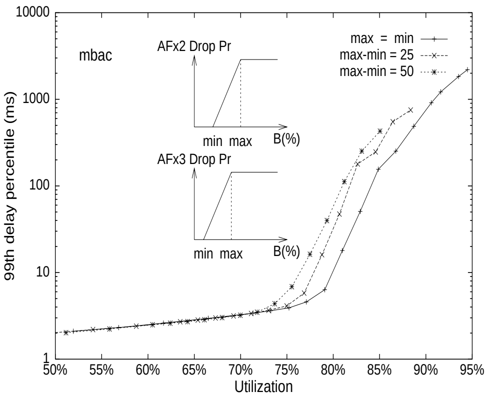

Figure 4. Offered load: 650%; source PCR: 20 to 80 Kbps; sources with PCR greater than 40 Kbps use AFx3 probes

The performance of different threshold settings has been evaluated also in figure 2. This figure differs from the previous one in two fundamental aspects: firstly, the offered load has been set very high (650%); secondly, different throughput values have been obtained by suitably traslating the thresholds, while maintaining their relative distance constant. Clearly a given setting performs better when, for a same carried load, the delay is lower.

In figures 3 and 4, the same performance investigation is carried out considering sources with PCR in the range from 20 to 80 Kbps. The AFx3 PROBE marking has been adopted for sources whose PCR is

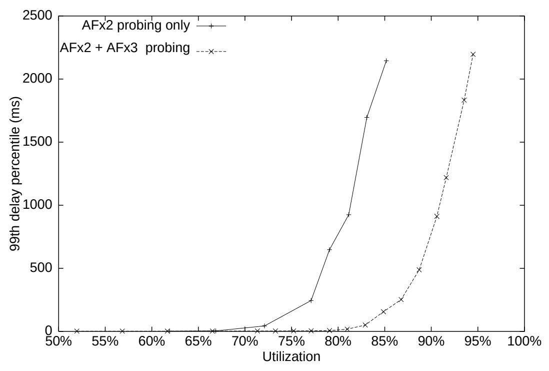

Figure 5. Comparison between AFx2 probing and AFx3+AFx3 probing with spreaded PCR distribution (20-80 Kbps) and very high offered load (650%)

greater than 40 kbps. Figure 3 reports results for various AFx2 dropping profiles configured with the *min* and *max* thresholds indicated in the label. The AFx3 dropping profile has been kept fixed with parameters $min = 50\%, max = 95\%$. In figure 4 the AFx3 dropping profile is obtained from the one of AFx2, by shifting back the *min* and *max* thresholds of 10%. Results are very different from the corresponding figure 2, and show that it is preferable to set the minimum threshold equal to the maximum one. The reason stays in the augmented traffic variance, due to the PCR variability, which wastes the improved accuracy of the measurement-based operation when separate thresholds are adopted. Instead, by setting tight thresholds, in very high load (650% in figure 4), the probability that a source with high PCR gets accepted dramatically reduces, and thus the system achieves better performance.

Finally, the benefical effects coming from the use of a double level of probing, AFx2 and AFx3, is highlighted in figure 5. In this figure we consider a very high offered load and sources emitting at a PCR in the range from 20 to 80 Kbps. The leftmost curve in figure 5 considers a scenario in which AFx2 packets are used for every probing attempt. Conversely, the rightmost curve has been obtained by using AFx3 probe-packets for flows with PCR greater than 40 Kbps. Results show the clear superiority of this second approach.

4. The packet forwarding/dropping API

The discussion carried out in the previous sections has shown that, in the context of admission control support, new low-level packet forwarding/dropping mechanisms should be implemented at the network routers. Hence, there is the need for node-level programming interfaces

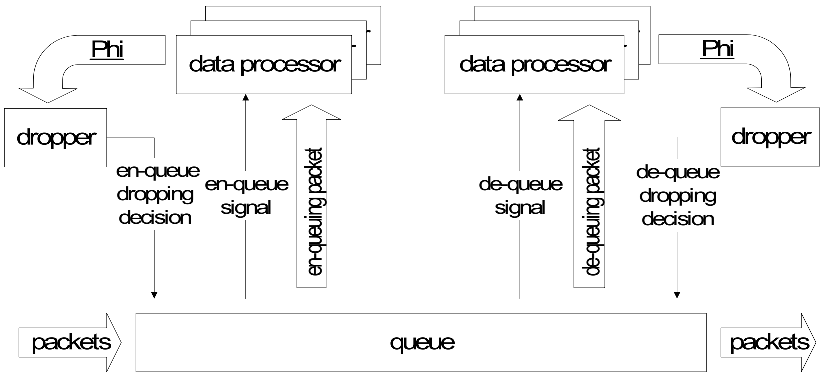

Figure 6. Architecture implementation. Three basic elements and their interconnection are shown: droppers, data processors, queues

that allow to build these mechanisms on top of the hardware router implementation.

To this purpose, we have developed a modular architecture, devised to allow the implementation of different packet-level dropping/forwarding mechanisms. Our architecture is composed of the following basic building blocks (figure 6): queue, data-processor, and dropper. Each module exposes an API that provides supplementary services. For reasons of space, in what follows we'll omit details about the specific commands supported by each API (additional documentation is available at the POLLENS project site, http://www.itea-pollens.org), while we'll focus our description on the module capabilities.

Queue module. Other than buffering packets, the queue provides asynchronous signals, i.e. events, that notify the arrival/departure of packets. These signals are generated in response to the corresponding en-queue (de-queue) packet request of the router operating system. A system module may register to the en-queue (de-queue) entry point, and, at each packet arrival (departure), obtain for each event, supplementary information (such as packet size, specific header fields, up to the whole packet payload).

Dropper. The dropper module provides dropping/forwarding decisions for arriving/departing packets. The architecture of the dropper is illustrated in figure 7. A dropper is bound to the en-queue (or de-queue) entry point of a queue. When an en-queue (de-queue) signal is generated, this signal is passed to the dropper, which responds with an "action list". This can be as simple as a packet dropping/forwarding decision, or a complex list of actions (e.g. "forward the current packet,

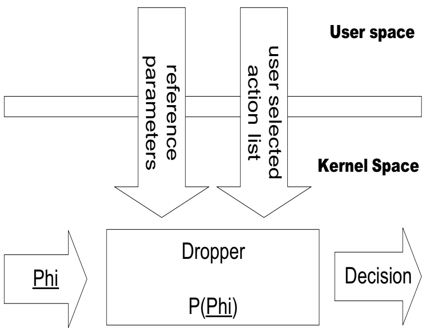

Figure 7. Dropper architecture

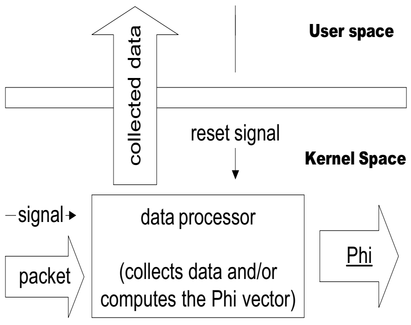

Figure 8. Data Processor architecture

and drop the one at the head of the queue"). An action list is selected based on:

- the default available action lists - every dropper has a built-in default action list. The minimal set of action lists is drop-packet and forward-packet;
- the user selected action list. At module instantiation, or at run time, the user can select a different set of actions for the dropping and/or forwarding decision;
- state information based on measurements taken on the system, and provided by a data processor bound to the dropper - since the state information is, in general, multidimensional, such an information will be passed via a vector Phi, where $Phi = phi_1, phi_2, phi_3, ., phi_n$, with n arbitrary;
- reference parameters - such as thresholds, probabilities, etc.

Data processor. We have called data processor the module that manages measurements and system states. A data processor uses the enqueue and/or de-queue signals to collect a number of statistics: offered load (packets/seconds, bytes/second); carried load; queue occupation. A supplementary task of the data processor is to filter the collected running statistics, where the filter parameters can be eventually modified during run-time operation. As shown in figure 6, note that more than one data processors can be instantiated on the same queue (for example, to provide measures at different time scales). Similarly, it can be useful to connect one data processor to more than one queue, to correlate events coming from them, as well as relate measures.

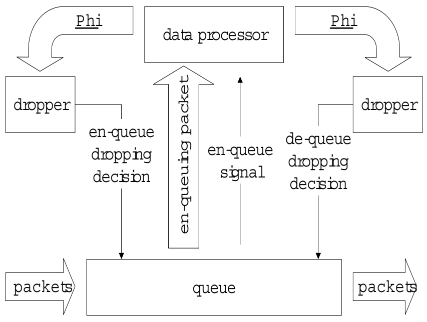

Figure 9. A simple grip implementation

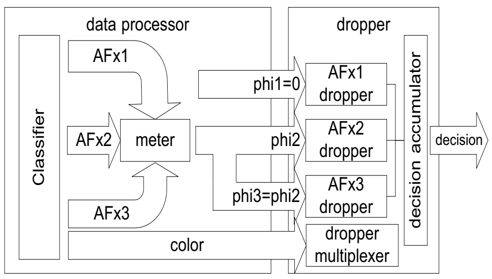

Figure 10. A closer look at the data processor and dropper implementations

The architecture of the data processor is shown in figure 8. While the role of the dropper is completely realized into the kernel space, i.e. it is a pure element that have been moved out by the modularization process of the queue, the data processor provides the capability of exposing the collected and processed data up to the user space. In particular, it may be essential to expose suitable traffic measurements and filtered queue congestion state to external applications (middleware applications, management applications, etc). This can be done easily by binding a data processor to the en-queue/de-queue event, without connecting it to any dropper. In this case we distinguish this class of data processors as the "passive" ones (since they do not take part into the decision process).

4.1. Examples of traffic control schemes implementations

Traffic management mechanisms can be implemented by suitably binding the described modules together[2].

A mechanism is implemented by first instantiating a mandatory queue module, and then binding eventual data processors and droppers. The dropping decision can be taken only fetching the dropper decision accumulator, in case one dropper is bound to the en-queue (de-queue) event. In particular the event sequence is: i) the en-queue (de-queue) packet request has been received by the queue; ii) the packet is notified to the en-queue (de-queue) registered data processors; iii) the data processors

[2]While modularity allows reusing already developed modules, as well as changing the forwarding behavior by simply composing modules in a different manner, the introduced flexibility has a price in terms of code optimization. A goal of the ongoing research activity is to understand the scalability limits of the proposed software platform.

elaborate the state vector Phi, and send it to the registered droppers (for each data processor); iv) the dropping decision is taken and enforced on the corresponding packet (i.e. the dropper is responsible to respond to the fetch_decision request generated by the queue, which then takes the specific dropping/forwarding action).

In order to illustrate the described operation, we now show how to implement the GRIP router described in section 2 with one data processor and two droppers (two instantiations of the same dropper template). We recall that the router measures AFx1 traffic, and drops AFx2 and AFx3 packets according to a piece wise linear dropping profile. Figure 9 shows the proposed GRIP architecture. The data processor has the task of measuring the incoming traffic, before it is actually offered to the queue (part of this traffic might be lost during congestion). The dropping actions are taken by the en-queue/de-queue droppers according to the Phi status information provided them. During congestion, they drop both AFx2 and AFx3 packets entering the queue, as well as AFx2 and AFx3 packets exiting the queue.

The role of the Phi vector, in this specific case, is to select the right dropping function via the color parameter, and refresh the decision stored in the decision accumulator. As shown in figure 10, the traffic data collected and filtered by the data processor are sent to the dropping functions via the vector parameters, *phi*2 and *phi*3. We set *phi*3=*phi*2 due to the simple fact that we want to discard the probe packets simply setting a more aggressive threshold in the AFx3 case (see section 3). Finally, note that AFx1 packets should never be dropped, and thus the dropper is fed with a constant state parameter fictitiously meaning that no congestion is encountered (e.g. *phi*1=0).

5. Conclusions

In this paper, we have shown that the problem of provisioning a DiffServ domain can be transformed into the problem of configuring packet discarding algorithms in core routers, in conjunction with the support of a distributed admission control function based on probing (GRIP). We have evaluated the performance of such an admission control function for various router configurations. An important result is the understanding that, when probes originating from different flows are mapped on different drop levels, significant performance enhancements can be achieved.

The application of packet dropping mechanisms to the new context of admission control requires the deployment of new packet-level traffic control functions, very different from the traditional approaches based on RED/RIO queues. To this purpose, we have proposed a packet forwarding/dropping API, based on three components: queue, droppers, and data processors.

Further research issues include the understanding whether the overhead, given by the introduction of the modularization process, will lead to scalability problems in the routers. Also, the integration of the described functionalities in a real-time video-conference test-bed is object of work in progress, in the frame of the European project POLLENS.

References

[1] S. Blade, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, "An Architecture for Differentiated Services", RFC2475, December 1998.

[2] S. Floyd and V. Jacobson. "Random Early Detection Gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking, vol. 1, no. 4, pp. 397-413, August 1993.

[3] S. H. Low and D. E. Lapsley, "Optimization Flow Control, I: Basic Algorithm and Convergence", IEEE/ACM Transactions on Networking, 7(6):861-75, Dec. 1999

[4] S. Floyd, T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm" , RFC2582, April 1999

[5] J. Heinanen, F. Baker, W. Weiss, J. Wroclavski "Assured Forwarding PHB Group", RFC 2597, June 1999.

[6] Kenjiro Cho, "The Design and Implementation of the ALTQ Traffic Management System.", dissertation, Keio University. January 2001

[7] $http : //diffserv.sourceforge.net$, Differentiated Services on Linux

[8] G. Bianchi, N. Blefari-Melazzi, "Admission control over assured forwarding PHBs: a way to provide service accuracy in a DiffServ framework", IEEE Global Telecommunications Conference (GLOBECOM) 2001, San Antonio, Texas, November 2001, pp. 2561-2565.

[9] G. Bianchi, N. Blefari-Melazzi, V. Mancuso: "Endpoint Admission Control over Assured Forwarding PHBs and its performance over RED implementations", LCNS 2170 - "Evolutionary trends of the internet", Proc. of IWDC 2001 Conference, Taormina,Italy,September 2001

[10] W. Almesberger, T. Ferrari, J.Y.Le Boudec: "SRP: a Scalable Resource Reservation Protocol for the Internet", IWQoS98, Napa (California), May 1998.

[11] L. Breslau, E. W. Knightly, S. Schenker, I. Stoica, H. Zhang: "Endpoint Admission Control: Architectural Issues and Performance", ACM SIGCOMM 2000, Stockholm, Sweden, August 2000.

[12] F.P. Kelly, P.B. Key, S. Zachary: "Distributed Admission Control", IEEE Journal on Selected Areas in Communications, Vol.18, No.12, December 2000.

[13] S.Jamin, P.B.Danzig, S.Shenker, L.A.Zhang, "A measurement-based admission control algorithm for integrated services packet networks", Trans. on Networking Vol. 5, No. 1, Feb 1997, pp. 56-70.

[14] G. Bianchi, V. Mancuso, G. Neglia: "Is Admission-Controlled Traffic Self-Similar?", Networking2002 Conference, May 2002, Pisa, Italy.