Accommodating Short and Long Web Traffic Flows over a DiffServ Architecture

Salvador Alcaraz[1], Katja Gilly[1], Carlos Juiz[2], and Ramon Puigjaner[2]

[1] Miguel Hernández University,
Departamento de Física y Arquitectura de Computadores,
Avda. del Ferrocarril, 03202 Elche, Spain
{salcaraz,katya}@umh.es
[2] University of Balearic Islands,
Departament de Ciències Matemàtiques i Informàtica,
Carretera de Valldemossa, km 7.5, 07071 Palma de Mallorca, Spain
{cjuiz,putxi}@uib.es

Abstract. DiffServ architecture has been widely used to achieve QoS over the Internet. Taking into account that HTTP traffic is the most extended protocol over the Internet community, many solutions have been proposed to supply QoS to this protocol. Traditionally, DiffServ architectures have considered two-colour markings in order to distinguish between high and low priorities. We investigate the special treatment for web traffic, whose pattern is very close to mice and elephants distribution flows in Internet. We differentiate flows into short and long classes in order to ensure QoS for short flows, but we try to achieve certain QoS for some long flows. Metering, shapering and marking processes are used to classify the incoming flows at the DiffServ using three-colour marking. The final algorithm has been named Long Flow Promotions (LFP). The simulation tool used is *ns2* and the realistic synthetic web traffic has been generated with *PackMime-HTTP*. The results are compared to RED and DropTail queue management. LFP gets reasonably low latency values while providing high priority level to short flows and improving some performance parameters such as overhead and dropped packets.

Keywords: web traffic, DiffServ, token bucket, QoS, short and long flows, packet promotion.

1 Introduction

Since the World Wide Web (www) was developed by Tim Berners-Lee [1] working at CERN, in Geneve, the Hypertext Transfer Protocol (HTTP) has been the communications protocol most widely used in Internet [2]. Recently, new applications (kazaa, P2P, etc.) and features (web 2.0) have been added to the Internet traffic, nevertheless the latest studies [3] show that web traffic is still the most usual data flow in Internet. At the early stages, the web traffic was composed of static and small pages, that used to contain a few objects. Later, database queries, dynamic pages and some ad-hoc objects based on flash technology were

N. Thomas (Ed.): EPEW 2011, LNCS 6977, pp. 14–28, 2011.

added to web traffic, that meant an increase in the web pages size and, hence, more packets per flow. Nowadays, web traffic implies that many technologies have to act together and interconnect the web around the world. Although HTTP does not provide any Quality of Service (QoS), different web users share the available bandwidth and the network resources of the Internet Service Provider (ISP). In this context, small web pages requested from web clients coexist with video streaming and database queries. The difference of size between each kind of flow can be considerable. If short flows are treated preferentially against long flows, some web flows could be excessively penalised or suffer considerable delay from server to client.

It is well documented that most of the Internet flows (around 80%) carry a short amount of traffic (around 20%), while the rest of flows (around 20%) represent most of the traffic (around 80%). These types of flows are named *mice* and *elephants*[4]. This fact sometimes leads to long response times for short browsing requests when the bandwidth is mostly used by long flows. Therefore, ISPs need to implement mechanisms to incorporate some enhanced QoS to their web sites in order to permit clients fast browsing without an excessive penalisation to rest of the flows. Regarding this subject, many solutions, environments and policies have been proposed [5].

Every application protocol in Internet generates a different traffic workload, but they might share the same First In First Out (FIFO) queue at the switching and routing nodes. If queues are allowed to drop packets only during overflow conditions, then bursty traffic flows will face greater dropping probabilities than smooth traffic flows [6]. Random Early Detection (RED) [7] has been one of the most important solutions to detect and avoid the congestion in computer networks. With RED queue management, packets are dropped with a certain probability before the queue reaches an overflow state. The main advantage of RED over TailDrop queue was analysed by [8]. RED queue operates with a lower queue size, especially during peak load and congestion conditions. This feature allows bursts of packets to be accommodated into the available queue achieving an overall performance improvement.

The remainder of this paper is organised as follows: section 2 describes our proposal: the Long Flow Promotion algorithm (LFP). Section 3 presents a comparative of the simulation results of the LFP, RED and DropTail algorithms. Finally, some concluding remarks are presented.

2 Long Flow Promotion (LFP)

Traditional QoS strategies are mainly based on marking and differentiating flows over the Differentiated Services Model (DiffServ), which has two dedicated devices: *edge* and *core*. Edge nodes mark the packets by adding different labels to them. The information contained in these labels specifies the workload conditions related to the Service Level Agreement (SLA) contracted by the client. When the marked packets reach the *core* node, they are forwarded over different queues by applying the suitable Active Queue Management (AQM) or a

stochastic treatment in order to achieve the required QoS. This paper presents the LFP algorithm as an approach to improve the web traffic over a DiffServ architecture focused to permit the coexistence between short and long flows in the same shared network.

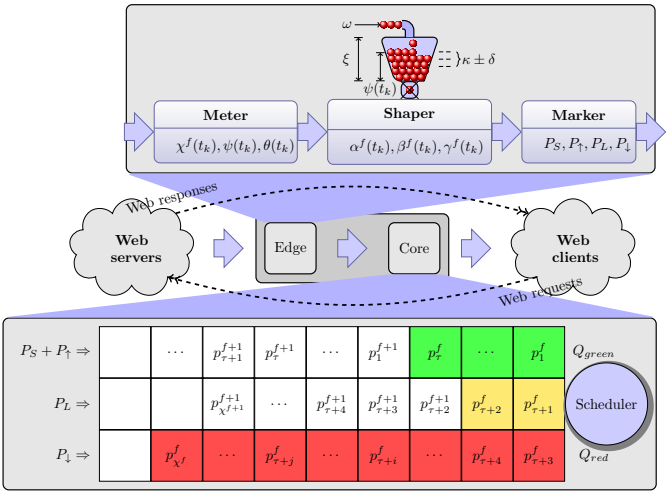
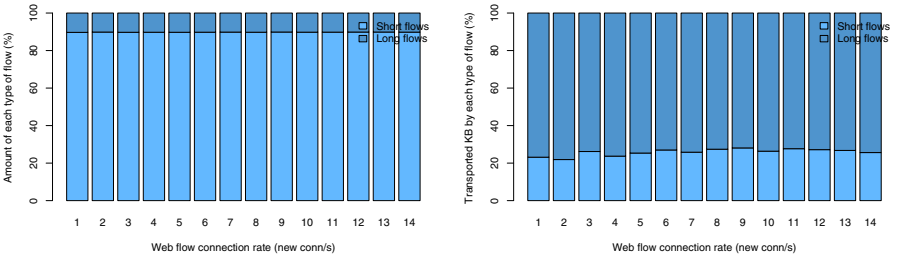


Fig. 1. LFP architecture. The DiffServ model is composed by the edge and core devices (in the middle). Edge device is composed by the meter, shaper and marker functions (at the top) and core device (at the bottom) is composed by the set of queues $Q = \{Q_{green}, Q_{yellow}, Q_{red}\}$ managed with priority queue scheduling algorithm.

Since web traffic has its own features, the LFP algorithm has been designed to improve the QoS of this type of traffic. The algorithm has been developed taking into account a couple of premises related to web traffic: a) End-users expectations; b) Mice and elephants paradigm.

a) The issue of *End-users expectations* has been widely discussed [9], and it is an important factor to determine the success of a website. When users are browsing the Internet, they want to go as fast as they can. Users can tolerate a long delay downloading heavy files such as multimedia streams, database queries or large documents, but they might not stand long delays while surfing the web; i.e. clicking into links or downloading small files such as images, sounds or any other small object. For these reasons, the end-users expectation should be strongly considered in a website development, and it is recommended to implement a suitable mechanism to improve the end-users perception about latencies and delays.

b) Web traffic flow size follows a well defined heavy-tailed [10] which is directly related to the *mice and elephants paradigm*. As we have commented previously, most of the traffic (around 80%) is carried out by a few flows (around 20%), that are denominated *elephants* in Internet. Therefore, the rest of flows, that are



(a) Web traffic flows (b) Web traffic data

Fig. 2. Mice and elephants flows and amount of transported data

named *mice*, transport around 20% of data information. This means that the most usual flows in the Internet are short flows, although they only transport a few bytes. Several studies of Internet traffic and packet distributions of the most representative protocols in Internet have been undertaken striving to establish the *threshold* (τ) that would differentiate between *short flows (SF)* and *long flows (LF)* in web traffic. Chen et al. [11] proposed a table with five representative types of web pages. The average of the web page size varies in the range of $[9, 12]$ Kbytes, from a minimum limit in the range of $[1, 3]$ Kbytes, until maximum values in the range of $[80, 90]$ Kbytes.

Considering an average size of 12 Kbytes, and using a Protocol Data Unit (PDU) of 1500 bytes, including an overhead of 60 bytes, and two ACK packets for the Transport Control Protocol (TCP) three-way handshake, we consider in our proposal a flow size average of 10 packets per flow as the threshold to differentiate between short and long flows. Since τ has been established, it is now possible to differentiate between short and long flows. *SF* will be those flows whose number of packets is lower than or equal to τ and therefore, *LF* are the rest of the flows. Fig. 2(a) shows a *mice/elephants* distribution for web response traffic. As it is depicted in the figure, around 90 % of flows are classified as *mice/SF*, therefore, the rest of the flows, around 10% are classified as *elephants/LF*. According to the *mice and elephants paradigm*, the amount of Kbytes transported for each type of flow is shown in Fig. 2(b). Using the same threshold, $\tau = 10\, packets$, around 20% of the whole web traffic in Internet is transported by short flows *(mice/SF)*, and the remainder of Kbytes, around 80%, are transported by the other type of flows *(elephants/LF)*. We are going to describe now our proposal for promoting flows in order to avoid unnecessary delays for the end-users.

2.1 Preferential Treatment for Short Flows (S_1)

Following with the above premises, the overall incoming traffic that reaches the QoS system should be measured and classified in order to assign the desired QoS level to the most sensitive web traffic, that are the short flows. As it is illustrated in Fig. 1, LFP is developed over a DiffServ architecture composed of two types of devices: *edge* and *core*. LFP mechanism is defined as follows: web

traffic from web servers (HTTP responses) reaches the DiffServ area where it is firstly measured and classified at the *edge* device. In this device, after going through the *meter* and the *shaper* process, the incoming packets are marked with different labels. When the packets leave the *edge* device and reach the *core* device, they are sent over a specific queue, depending on the assigned label. Finally, the priority queueing scheduling strategy at the *core* device configures the QoS level in the system.

The overall incoming traffic that reaches the system is divided in n flows and defined as $f_1, f_2, ..., f_n$. Each flow is composed of a sequence of p packets. Therefore, the flow f contains p packets and the packets sequence is defined as: $p_1^f, p_2^f, ..., p_p^f$, where p_i^j defines the i-packet from the flow j, and that arrives to the system at the instant t_i^j, as it is depicted in Fig. 3.

Let us define $P^f = \{p_i^f, \forall i \in [0, t_k]\}$ as the set of packets from flow f. Therefore, $V(t_k) = \{P^f \mid \forall f\}$ is the overall traffic at the interval $[0, t_k]$. In order to simplify the expressions, several assumptions have been taken:

- The amount of data from each flow is quantified as packets instead of Kbytes.
- Only one packet reaches the system in the interval $[t_{k-1}, t_k)$.
- The web traffic considered corresponds only to responses. The request sizes are negligible compared to the response sizes.

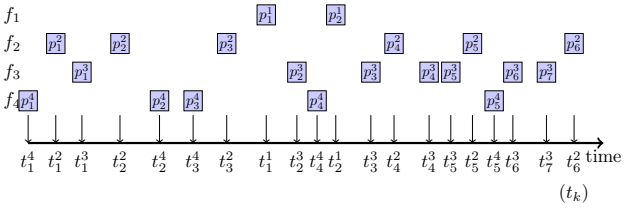


Fig. 3. Example of packet sequence that reaches the QoS system. Packets are labelled as follows: p_i^f is the *k-th* packet from flow f, and it arrives at instant t_i^f.

Let us now describe the functions in the *edge* device. The *meter* is the first function that is computed, $\chi^f(t_k)$, and represents the number of packets of the flow f at time $[0, t_k]$ (i.e. from its birth until t_k). It is defined as follows:

$$\chi^f(t_k) = Ord(P^f) \tag{1}$$

After the *meter* process, the packets go through the *shaper* function, that defines the transition of a flow from short to long state, and it is defined by the threshold τ. First of all, the differentiation condition, S_1:

$$S_1 \equiv \chi^f(t_k) \leq \tau \tag{2}$$

The *shaper* function at the *edge* node is defined for each incoming flow f at instant t_k as the discrete function $\alpha^f(t_k) \in \{0, 1\}$, defined as follows:

$$\alpha^f(t_k) = I(S_1), \tag{3}$$

where the discrete function $I(S)$ is:

$$I(S) = \begin{cases} 1 & \text{if S is true} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

2.2 Packets Promotion Close to the Threshold (S_2)

Considering only the S_1 constraint to differentiate between short and long flows, the consequences for a flow f that arrives to the DiffServ system, will be the following:

- The range of packets $[1, \tau]$ receives always the highest priority level.
- The range of packets $[\tau + 1, \chi^f(t_k)]$ receives always a low priority level.

Hence, S_1 establishes a hard threshold between short and long flows. Therefore, flows of $\tau + i$ packets, where $i = 1, 2, 3...$ are considered as long flows, despite their size being close to τ. Web traffic flow size distribution follows a heavy-tailed distribution, that is, there are many flow sizes close to τ. For this reason, the constraint that we define in this section tries to improve the QoS parameters for those flows with sizes close to the threshold τ, by introducing the *packets promotion* concept. This concept is related to those flows that are a bit longer than the threshold (τ), and that under particular system conditions, could be considered as short flows and, hence, receive a high priority QoS level. The most appropriate conditions to promote packets are low congestion and idle state of the system.

To deploy the S_2 constraint, the Token Bucket Model [12] has been used to detect the idle system state by counting the packet promotions. Although, the packet promotion is restricted in order to prevent either the increase of the system overhead or the promotion of the inappropriate packets.

The token bucket model is used to compute the amount of packets than can be promoted. The token bucket operation is defined as follows: the bucket emulates a depot of tokens, where each token indicates the possibility to send a packet over the high priority queue. The maximum capacity of the token bucket is defined by ζ tokens. New tokens are supplied at ω ratio in $tokens/s$. Tokens are always consumed with packets forwarded over the high priority queue, therefore, packets from short flows and promoted packets from long flows. The physical token bucket capacity is limited by a low and high level. If the tokens level is below the low level, it is considered that no tokens are available at the bucket. Otherwise, if the tokens level is higher than the high level, no more tokens can fill the bucket. The amount of available tokens in the bucket indicates the available bandwidth to offer a high priority service to incoming flows. If $V^*(t_k)$ defines the amount of promoted packets, then the state of the token bucket at instant t_k is defined by $\psi(t_k)$ as follows:

$$\psi(t_k) = \psi(t_{k-1}) + IN(t_k) - OUT(t_k) \tag{5}$$

where:

$$IN(t_k) = min(\omega * (t_k - t_{k-1}, \zeta - \psi(t_{k-1}))) \tag{6}$$
$$OUT(t_k) = V^*(t_k) I\{V^*(t_k) \leq (\psi(t_{k-1}) + min(\omega * (t_k - t_{k-1}), \zeta - \psi(t_{k-1})))\}$$

As the token bucket model is used to bound the quantity of promoted packets, $\theta(t_k)$ represents the normalised capacity of the bucket, defined as follows:

$$\theta(t_k) = \left[\frac{\psi(t_k)}{\zeta}\right]_0^{100} \tag{7}$$

However, it is desirable that the state of the token bucket, $\psi(t_k)$, remains close to a precise level or set point defined by $\kappa \in [0, 100]$ as it is depicted in Fig. 1. As the web traffic presents peaks of incoming traffic, $\psi(t_k)$ must range around a set point κ. In order to permit this working area, top and bottom limits are defined by the parameter $\delta \in [0, 100]$. Therefore, the working area, that is defined by $\kappa \pm \delta$ operates as follows:

- If $\psi(t_k)$ is close to κ, the packets promotion is at *open* state.
- If $\psi(t_k)$ reaches $\kappa - \delta$ level, the packets promotion will turn to *close* state. At this point, no packets are promoted. Hence, tokens are only consumed with packets from short flows. As the token bucket continues being filled with new tokens at ω ratio, then the token bucket level $\theta(t_k)$ will catch up $\kappa + \delta$ level again. At this point, the token bucket state will turn to *open* state again and the promotion process will be reactivated.

The token bucket operation bounds are defined by $[\kappa - \delta, \kappa + \delta] \mid 0 \leq \kappa - \delta \leq \kappa + \delta \leq 100$. Considering the above bounds, the differentiation function S_2 can be defined as follows:

$$S_2 \equiv \left\{\theta(t_k) \geq \kappa + \delta\right\} \vee \left\{(\kappa - \delta \leq \theta(t_k) \leq \kappa + \delta) \wedge \beta^f(t_{k-1})\right\} \tag{8}$$

Once the metering process has concluded, the shaper process computes $\beta^f(t_k)$ function for each incoming flow:

$$\beta^f(t_k) = I(S_2) \tag{9}$$

S_2 alleviates S_1 weakness by using the packets promotion, but an adverse effect appears under particular circumstances of low congestion: when there are a few flows crossing the system, the token bucket state could be promoting some inappropriate flows. These flows can be extremely long, and they do not need any higher priority level because the end-users expect a long delay for these flows any way.

2.3 Detecting Elephant Flows (S_3)

The presence of *elephant* flows in the QoS system is always bad news, and it produces an overall system performance fall. For that reason, the main goal of

the constraint S_3 is the detection and isolation of extremely long flows. Such flows are classified as *elephant*, hence, they do not need some QoS requirements, and they can be treated with the lowest priority.

In order to detect and isolate those very long flows, the critical issue is to define the measurement to be applied. As the web traffic nature is very variable (heavy tail and self-similar distributions), setting a threshold as a fixed number of packets to differentiate flows as long or very long is not suitable, because an excessive low threshold could generate too many promoted packets and, by contrast, an excessive high threshold could generate too few promoted packets. None of both circumstances are desirable. For this reason, the value to determine when a flow is long or very long must be adaptive to the traffic conditions of each situation. Hence, we consider that it has to be calculated from the sizes of the last flows that have crossed the QoS system. Therefore, $X_{t_k,H}$ is the set of the last H flow sizes that have passed through the system, and is defined as follows:

$$X_{t_k,H} = \{\chi^f(i) \mid i \in [t_{k-H}, t_k], H \in \mathbb{N}, \forall f \mid P^f \in V(t_k)\} \tag{10}$$

Considering $F_{X_{t_k,H}}(x)$ as the distribution function of $X_{t_k,H}$, the $u - quantile$ over (10) is defined by $Q_{X_{t_k,H}}(u)$ as follows:

$$Q_{X_{t_k,H}}(u) = Inf\{x \mid F_{X_{t_k,H}}(x) \geq u\} \tag{11}$$

$Q_{X_{t_k,H}}(u)$ is a dynamic and variable flow size measurement of the recent history of the flows crossing the system. As the S_3 goal is the detection and isolation of those flows whose sizes are extremely long, or longer than the last flows on the system, then the S_3 function is computed from (11) as follows:

$$S_3 \equiv \chi^f(t_k) \geq Q_{X_{t_k,H}}(u) \tag{12}$$

With the above definitions, S_3 is the differentiating condition between *elephants* and just long flows. In this way, flows longer than $Q_{X_{t_k,H}}(u)$ are considered as *elephants*, therefore, they should be considered with the lowest priority level. In the other case, they are considered as flows with medium priority level. By applying (4) to the differentiation function, the function $\gamma^f(t_k)$ is added to the *shaper* module:

$$\gamma^f(t_k) = I(S_3) \tag{13}$$

2.4 Scheduling the Packets

The last process at the edge device is the *marker*, which uses the set of labels $L = \{P_S, P_L, P_\uparrow, P_\downarrow\}$ for marking the incoming packets with a label $l \in L$ according to the following rules:

$$l = \begin{cases} P_S & \alpha^f(t_k) \\ P_L & \overline{\alpha^f(t_k)} \wedge \overline{\beta^f(tk)} \wedge \overline{\gamma^f(tk)} \\ P_\uparrow & \overline{\alpha^f(t_k)} \wedge \beta^f(t_k) \wedge \overline{\gamma^f(tk)} \\ P_\downarrow & \overline{\alpha^f(t_k)} \wedge \overline{\beta^f(t_k)} \wedge \gamma^f(t_k) \end{cases} \tag{14}$$

From the above expressions, the LFP algorithm is modelled as the finite state machine depicted in Fig. 4. LFP is composed by the set of states $Q = \{q_0, q_1, q_2\}$ and the set of transitions $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$, where $t_1 = \alpha^f(t_k)$, $t_2 = \overline{\alpha^f(t_k)}$, $t_3 = \overline{\gamma^f(t_k)} \wedge \overline{\beta^f(t_k)}$, $t_4 = \overline{\gamma^f(t_k)} \wedge \alpha^f(t_k)$, $t_5 = \gamma^f(t_k)$ and $t_6 = 1$.

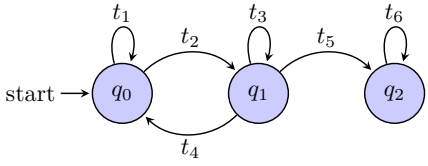


Fig. 4. Finite State Machine of LFP algorithm

When the packets have left the *edge* device, they reach the next device at the DiffServ architecture, that is the *core* device as it is illustrated in Fig. 1. This device is defined with the set of queues $Q = \{Q_{green}, Q_{yellow}, Q_{red}\}$. The packets marked as P_S or $P_\uparrow$ are forwarded over Q_{green}; packets marked with P_L are forwarded over Q_{yellow} and finally, packets marked with $P_\downarrow$ are forwarded over the lower priority queue Q_{red}. The dispatching algorithm based on priority queuing where Q_{green} is the highest priority queue, Q_{yellow} is the intermediate priority queue and finally, Q_{red} as the lowest priority queue. Hence, the highest QoS is assured for P_S and $P_\uparrow$ packets. The penalisation is for $P_\downarrow$ packets because they are always forwarded over Q_{red}. And the rest of them, P_L packets, receive intermediate QoS level, as they neither are *elephants*, nor have received a high priority QoS level due to incoming traffic conditions, token bucket configuration or flow length.

The election of a suitable $u-quantile$ over $X_{t_k,H}$ is important for establishing the threshold of *elephants* detection. Experimental values are obtained from the simulation results that are shown in Fig. 5. Let us considerer $u = 99$, then we get a value of $v = 123\,packets$ and only 20% of the long flows are marked as *elephants*. A 90-quantile of $X_{t_k,H}$ gets a value of $v = 17\,packets$, and 35% of long flows are marked as *elephants*. We have decided to select an intermediate value, the 95-quantile, that means $v = 30\,packets$. In this case, 30% of long flows are marked as *elephants*.

3 Simulation Results

The simulation has been driven using ns2 [13] in order to improve the end-to-end web traffic latency and analyse the effect over other performance parameters such as dropped packets, throughput and overhead. The network architecture used is based on a single bottleneck dumbbell topology where the DiffServ model has been implemented (see Fig. 1). The QoS system has been implemented as a bottleneck link of 2 Mbps in order to appreciate the congestion level produced by the incoming traffic. Some authors [14,15] have recommended the use of small

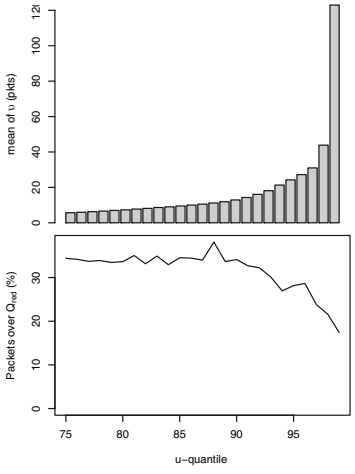


Fig. 5. Effect of u-quantile over the *elephants* detection and υ value

buffers in internetwork devices, therefore, according to these recommendations, every device buffer has been configured with a capacity of 50 packets. Parameters related to the buffer size have been modified ($min_{th} = 10, max_{th} = 40$). Both *edge* and *core* devices at the DiffServ architecture have been configured as the RED management with the original values ($max_p = 0.02, w_q = 0.001$). Web traffic is the only traffic that goes through the system. Web clients are modelled as a cloud where the web requests go through the QoS system and reach the web server cloud. Web server responses return from the web server cloud and arrive to the clients after going through the system. For our purposes, web traffic requests are negligible and the analysis is focused only in web traffic responses. The incoming synthetic traffic has been generated by using HTTP PackMime [16], where the web traffic is modelled as stochastic models obtained from the traffic analysis of a real link. The web traffic intensity is modulated with the R parameter, that sets up the incoming traffic as *new conn/s* in the system. We run 10 simulations with the same parameter values but varying the seed, and the results obtained are averaged in order to achieve a higher degree of confidence. In order to analyse the effect of the congestion, three congestion levels have been considered and summarised in the Table 1: *low*, *medium* and *heavy*.

After the marking process at the *edge* node and the scheduling packets process at the *core* node, the packets distribution among Q_{green}, Q_{yellow} and Q_{red} queues is depicted in Fig. 6. As it has been mentioned above, the first τ packets from every flow are marked with label P_S, and therefore, are forwarded over Q_{green}. From $\tau + 1$ packets onward every flow is considered as a long flow. Depending on $\beta^f(t_k)$, some packets are marked with $P_\uparrow$ and therefore, they are forwarded over Q_{green} as well. In the same way, after applying $\gamma^f(t_k)$, elephant flows are detected, and therefore, their packets are marked with $P_\downarrow$ and forwarded over Q_{red}. Finally, the rest of packets are marked as P_L and forwarded over Q_{yellow}.

Table 1. The R parameter has been used to establish each congestion level. The parameters related to the incoming traffic are been calculated : C, mean of simultaneous connections; F, mean simultaneous flows. Performance parameters related to the shared resource are: U, utilisation (%); *Throughput* in Mbps, is the amount of TCP data transmitted per time; *Goodput* in Mbps, is the amount of HTTP data transmitted per time; *Overhead* produced to transmit the required HTTP data; P^{drop} packet drop probability and F^{drop} is the probability that a flow has dropped packets.

Level			*Alg*	U	*Throughput*	*Goodput*	*Overhead*	P^{drop}	F^{drop}
R	C	F							
Low			DropTail	17.8	0.371	0.356	4.12	1.2e-03	4.08e-03
6	3.22	32.73	RED	18.1	0.383	0.361	5.66	1.56e-02	3.89e-02
			LFP	17.8	0.371	0.356	4.22	1.59e-03	3.33e-03
Medium			DropTail	31.4	0.657	0.628	4.50	6.12e-03	2.10e-02
10	4.81	52.84	RED	32.9	0.706	0.658	6.79	2.77e-02	7.15e-02
			LFP	30.9	0.647	0.618	4.51	4.82e-03	1.05e-02
Heavy			DropTail	46.8	0.986	0.936	5.15	1.60e-02	5.63e-02
14	6.91	73.68	RED	43.6	0.947	0.872	7.85	3.85e-02	9.77e-02
			LFP	43.3	0.912	0.867	4.94	9.43e-03	2.24e-02

Focusing on the end-to-end latency metric of a particular traffic level fixed by $R = 10\, conn/s$, the evolution of the latency over the flow size, is shown in Fig. 7. This figure shows that DropTail always gets the worst latency for every flow size. For the shortest flows, RED shows a slightly better behaviour than LFP, but from τ onward, the latency trend changes and its values for LFP are normally lower than RED. This can be explained because from the minimum flow size to τ, every flow is treated with the highest quality of service level in LFP, that clearly improves the latency versus DropTail. Meanwhile, the latency in this range compared to RED suffers a minimal penalisation because of the packets promotion. Close to the τ, all the packets from long flows are treated as short flows packets, and hence, obtain the same latency.

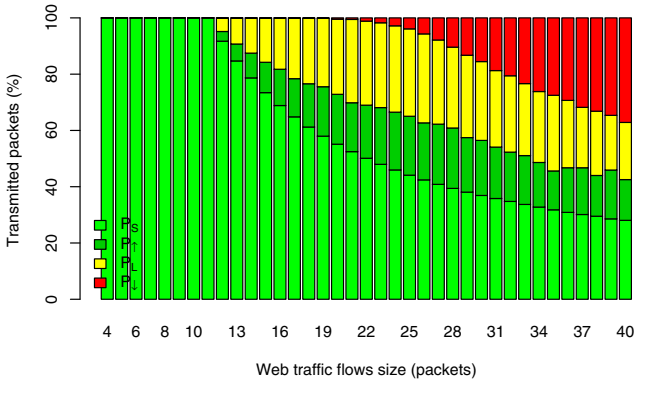


Fig. 6. Packet label classification

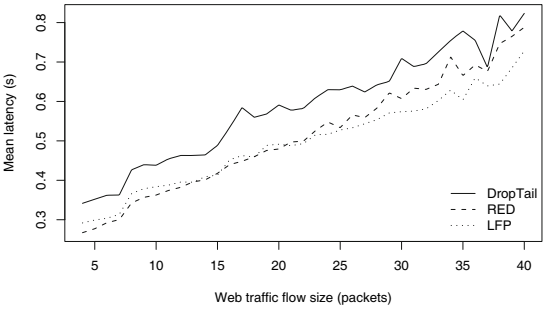


Fig. 7. End-to-end web traffic latency

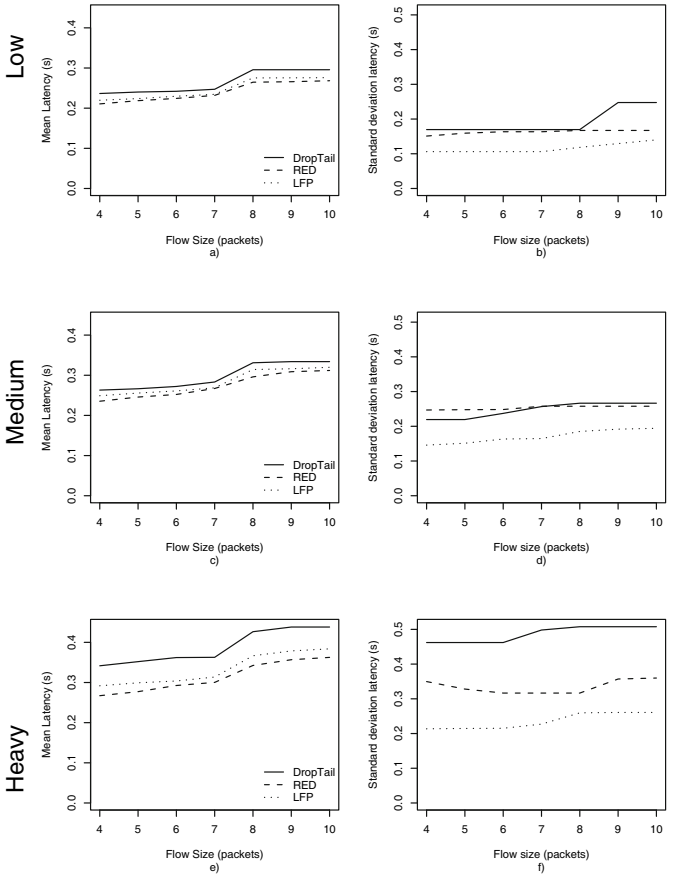


Fig. 8. Mean and standard deviation latency

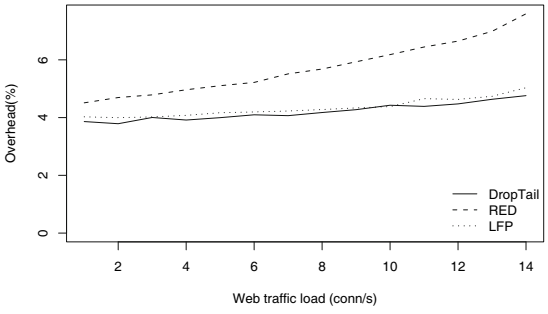


Fig. 9. Overhead

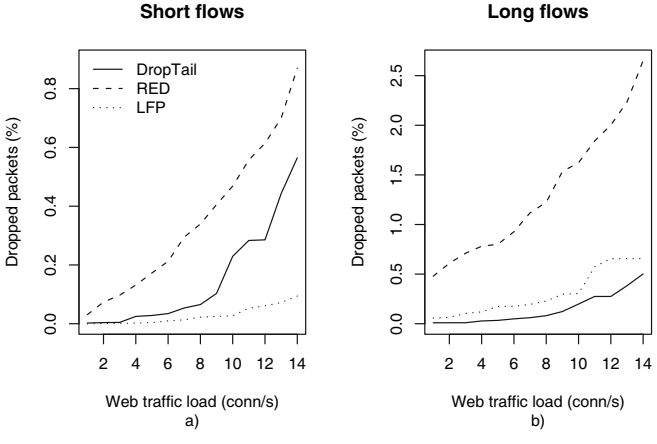


Fig. 10. Dropped packets

The latency of short flows has been isolated and plotted in Fig. 8. The mean latency and standard deviation for DropTail, RED and LFP, for each congestion level have been drawn. Regarding the latency, there are not substantial differences between them. The latency for LFP always remains between DropTail and RED. The standard deviation for LFP is always lower than the other two proposals.

The end-to-end final latency must be analysed with other performance parameters. As it has been summarised in Table 1, LFP obtains a considerably lower *overhead* than RED and very similar *goodput*, for every congestion level. Related to dropped packets, LFP gets the lowest P^{drop} and F^{drop} for each congestion level as well. The *overhead* evolution for each algorithm is clearly depicted in Fig. 9. While DropTail and LFP get almost constant *overhead* when increasing the web traffic load, RED shows a growing trend. Even for heavy congestion level, RED overhead reaches up to 8%, while DropTail and LFP remain slightly over 4%.

Dropped packets effect for DropTail, RED and LFP is shown in Fig. 10. For short flows, LFP is always the proposal with the fewest dropped packets for every web traffic scenario. Obviously, the short flows preferential treatment produces a growth in the long flow queue and therefore there are more dropped packets at heavy congestion level. However, the dropped packets phenomena for long flows using LFP is higher than DropTail, but it is lower than RED for every web traffic scenario.

4 Conclusions

DiffServ architecture has been placed as the most suitable environment to deploy QoS issues in the ISPs. As QoS is not implemented in HTTP protocol, we propose an algorithm named LFP to get preferential treatment for short flows. We also consider the promotion of some long flows under some circumstances and, finally, the penalisation of the extremely long flows.

RED and DropTail are popular algorithms that also implement QoS in a DiffServ environment. We have compared the ns2 simulation results obtained with ns2 for LFP, RED and DropTail. We have observed that LFP clearly outperforms DropTail and obtains similar results than RED in terms of mean latency, but improves its standard deviation. Considering the *overhead*, LFP shows an important improvement compared to RED for all congestion levels. There are also benefits in the packet drop probability as LFP always drops less packets than RED. Therefore we consider that LFP algorithm described in this paper is a suitable solution to be used in a QoS Diffserv architecture.

References

1. Berners-Lee, T., Fielding, R., Frystyk, H.: Hypertext transfer protocol – HTTP/1.0 (1996)
2. Cleveland, W.S., Sun, D.X.: Internet traffic data. Journal of the American Statistical Association 95, 979–985 (2000); reprinted in Raftery, A.E., Tanner, M.A., Wells, M.T. (eds.) Statistics in the 21st Century, pp. 214–228. Chapman & Hall/CRC, New York (2002)
3. Kim, H., Claffy, K., Fomenkov, M., Barman, D., Faloutsos, M., Lee, K.: Internet traffic classification demystified: myths, caveats, and the best practices. In: CONEXT 2008: Proceedings of the 2008 ACM CoNEXT Conference, pp. 1–12. ACM, New York (2008)
4. Guo, L., Matta, I.: The war between mice and elephants. In: Ninth International Conference on Network Protocols, pp. 180–188 (2001)
5. Firoiu, V., Le Boudec, J.Y., Towsley, D., Zhang, Z.L.: Theories and models for Internet quality of service. Proceedings of the IEEE 90, 1565–1591 (2002)
6. Crovella, M.E., Bestavros, A.: Self-similarity in world wide web traffic: evidence and possible causes. IEEE/ACM Trans. Netw. 5, 835–846 (1997)
7. Floyd, S., Jacobson, V.: Random early detection gateways for congestion avoidance. IEEE/ACM Trans. Netw. 1, 397–413 (1993)

8. Brandauer, C., Iannaccone, G., Diot, C., Ziegler, T., Fdida, S., May, M.: Comparison of tail drop and active queue management performance for bulk-data and web-like Internet traffic, p. 0122. IEEE Computer Society, Los Alamitos (2001)
9. Jun, C., Shun-Zheng, Y.: The structure analysis of user behaviors for web traffic. In: ISECS International Colloquium on Computing, Communication, Control, and Management, CCCM 2009, vol. 4, pp. 501–506 (2009)
10. Zhu, X., Yu, J., Doyle, J.: Heavy tails, generalized coding, and optimal web layout. In: Proceedings of Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM 2001, vol. 3, pp. 1617–1626. IEEE, Los Alamitos (2001)
11. Chen, X., Heidemann, J.: Preferential treatment for short flows to reduce web latency. Comput. Networks 41, 779–794 (2003)
12. Ahmed, N.U., Wang, Q., Barbosa, L.O.: Systems approach to modeling the token bucket algorithm in computer networks. Mathematical Problems in Engineering 8 (3), 265–279 (2002)
13. The network simulator NS-2, `http://www.isi.edu/nsnam/ns/`
14. Kelly, F., Raina, G., Voice, T.: Stability and fairness of explicit congestion control with small buffers. SIGCOMM Comput. Commun. Rev. 38, 51–62 (2008)
15. Shifrin, M., Keslassy, I.: Modeling TCP in small-buffer networks. In: Das, A., Pung, H.K., Lee, F.B.S., Wong, L.W.C. (eds.) NETWORKING 2008. LNCS, vol. 4982, pp. 667–678. Springer, Heidelberg (2008)
16. Cao, J., Cleveland, W., Gao, Y., Jeffay, K., Smith, F., Weigle, M.: Stochastic models for generating synthetic HTTP source traffic. In: INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, vol. 3, pp. 1546–1557 (2004)