Universitat de les
Illes Balears

DOCTORAL THESIS
2015

DEPLOYING AN IMPROVEMENT FOR WEB TRAFFIC QOS OVER DIFFSERV

Salvador Alcaraz Carrasco

Universitat de les
Illes Balears

DOCTORAL THESIS
2015

Doctoral Programme of Information and Communications Technology

DEPLOYING AN IMPROVEMENT FOR WEB TRAFFIC QOS OVER DIFFSERV

Salvador Alcaraz Carrasco

Thesis Supervisor: Dr. Carlos Juiz
Thesis Supervisor: Dra. Katja Gilly
Thesis tutor: Dr. Carlos Juiz

Doctor by the Universitat de les Illes Balears

Dedicated to my friend Javi, there where you are

Acknowledgments

I would like to thank Prof. Ramon Puigjaner, my thesis supervisors Dr. Carlos Juiz and Dr. Katja Gilly, and of course, Magdalena, Celia and Lucía.

Thesis supervisor
Carlos Juiz and Katja Gilly

Author
Salvador Alcaraz

Abstract

DiffServ architecture has been widely used to accomplish the QoS requirements in the Internet. Traditionally, the DiffServ framework includes two devices, *edge* and *core*. Regarding the processes, DiffServ usually needs a two colours marking process to characterise the incoming traffic in the *edge* device, and then, the scheduling process at the *core* device forwards the coloured packets to different queues attending to priority rules. Furthermore, web traffic continues being one of the most important protocols in Internet, and must coexist with other new protocols and applications. New proposals must be developed to enhance the QoS features of this type of Internet traffic. Following these trends, we investigate the differentiated treatment for web traffic. The web flow analysis shows that this type of Internet traffic is very close to the *mice and elephants paradigm*, well known in the Internet traffic. On the other hand, the user perceived Quality of Service is a deciding factor to achieve the success of certain website. Our proposal, named LFP (Long Flow Promotion), aims for the above features based on a DifFServ architecture. It uses a three colour set to differentiate the incoming traffic at the *edge* device. It also introduces the token bucket model as a traffic detection mechanism and packet promotion. Finally, the last implemented key is the extremely long flows detection and isolation, which is related to the packet penalisation, that focuses on enhancing the overall performance of the DiffServ system by managing the web traffic. The proposed algorithm, Long Flow Promotions (LFP), has been tested by simulation using *ns2*. It has been compared with other well-known proposals such as RED and DropTail and some performance parameters are been analysed, such as: latency, dropped packets, overhead and throughput. LFP gets reasonable values for the performance improvements we introduce in the algorithm compared to the other proposals.

Directores de tesis
Carlos Juiz and Katja Gilly

Autor
Salvador Alcaraz

Resumen

La arquitectura DiffServ ha sido utilizada para proporcionar los requerimientos de QoS en Internet. Tradicionalmente, el entorno DiffServ incluye dos dispositivos, denominados *core* y *edge*. En relación al proceso, DiffServ ha utilizado habitualmente dos colores en el proceso de marcado del tráfico entrante al dispositivo edge, y posteriormente, el dispositivo core ha retransmitido los paquetes coloreados por diferentes colas atendiendo a criterios de prioridades. Por otro lado, aunque el tráfico web continua siendo uno de los protocolos más importantes en Internet, tiene que coexistir con otros protocolos y aplicaciones nuevas. Por lo tanto, nuevas propuesta deben ser desarrolladas para mejorar las características de QoS sobre este tipo de tráfico. Continuando con estas tendencias, hemos investigado un tratamiento diferenciado para el tráfico web. El análisis del tráfico web muestra que este tipo de tráfico coincide mucho con el paradigma de las elefantes y ratones, bien conocido en Internet. Y por otro lado tenemos que, la Calidad del Servicio percibida por el usuario será un factor decisivo para conseguir el éxito de un website. Nuestra propuesta, denominada LFP (Long Flow Prommotion) está orientada a las premisas anteriores y desarrollada sobre la arquitectura DiffServ. Utiliza un conjunto de tres colores para la diferenciación de paquetes en el dispositivo edge. Además, introduce el modelo token bucket como mecanismo de detección y promoción de paquetes. Finalmente, la última propiedad en nuestra propuesta es la detección y aislamiento de flujos extremadamente largos, muy relacionada con la penalización de paquetes, todo ello enfocado a mejorar el rendimiento global del entorno DiffServ en el tratamiento del tráfico web. El algoritmo LFP ha sido evaluado e implementado en simulación utilizando *ns2*. Ha sido comparado con otras propuestas habituales como son DropTail y RED, y han sido analizados algunos parámetros de rendimiento como son: latencia, descarte de paquetes, sobrecarga y productividad. LFP obtiene un comportamiento razonable para algunos parámetros analizados y mejora algunos otros para diferentes escenarios de tráfico.

Directors de tesi
Carlos Juiz and Katja Gilly

Autor
Salvador Alcaraz

Resum

L'arquitectura DiffServ ha estat utilitzada per proporcionar els requeriments de QoS en Internet. Tradicionalment, l'entorn DiffServ inclou dos dispositius, denominats *core* i *edge*. En relació al procés, DiffServ ha utilitzat habitualment dos colors en el procés de marcat del tràfic entrant al dispositiu edge, i posteriorment, el dispositiu core ha retransmès els paquets acolorits per diferents cues atenent a criteris de prioritats. D'altra banda, encara que el tràfic web contínua sent un dels protocols més importants en Internet, ha de coexistir amb altres protocols i aplicacions noves. Per tant, noves proposta han de ser desenvolupades per millorar les característiques de QoS sobre aquest tipus de tràfic. Continuant amb aquestes tendències, hem investigat un tractament diferenciat per al tràfic web. L'anàlisi del tràfic web mostra que aquest tipus de tràfic coincideix molt amb el paradigma de les elefants i ratolins, ben conegut en Internet. I d'altra banda hem de, la Qualitat del Servei percebuda per l'usuari serà un factor decisiu per aconseguir l'èxit d'un website. La nostra proposta, denominada LFP (Long Flow Prommotion) està orientada a les premisses anteriors i desenvolupada sobre la arquitetura DiffServ. Utilitza un conjunt de tres colors per a la diferenciació de paquets en el dispositiu edge. A més, introdueix el model token bucket com a mecanisme de detecció i promoció de paquets. Finalment, l'última propietat en la nostra proposta és la detecció i aïllament de fluxos extremadament llargs, molt relacionada amb la penalització de paquets, tot això enfocat a millorar el rendiment global de l'entorn DiffServ en el tractament del tràfic web. L'algorisme LFP ha estat avaluat i implementat en simulació utilitzant NS2. Ha estat comparat amb altres propostes habituals com són DropTail i RED, i han estat analitzats alguns paràmetres de rendiment com són: latència, descarti de paquets, sobrecàrrega i productivitat. LFP obté un comportament raonable per a alguns paràmetres analitzats i millora alguns altres per a diferents escenaris de tràfic.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

When users are browsing the Internet, they are generating multiple types of flows depending on the websites that they are visiting or the data they are requesting. From the point of view of the generated pages size, we can find very different types of pages: small pages (some bytes), updating web 2.0 pages (some tens of bytes), medium size pages (some hundreds of bytes), database queries (some Kilobytes) and downloaded files (up to gigabytes). Although web users know that the download time for each type of flow will be different depending on the flow size, they expect to download it as fast as possible, and if they perceive an excessive delay in a certain website, probably, they will swap to other faster website. This issue may mean problems for a website as it will loose popularity. From the point of view of the quality of service, websites try to provide the fastest response in terms of delay and web user perception, as possible.

As we have mentioned above, the size of flows is very variable from very short to extremely long flows. Every flow of the website shares the same network resources: web servers, links, internetworking devices, etc, and it is necessary to establish some mechanism to distinguish each type of flow. Nowadays, the QoS implemented on the websites is developed over the DiffServ framework with the standard devices: edge and core. The first device, the edge, is placed as a border device and dedicated to classify the incoming packets by marking them with different colours. The second device, the core, is placed in the core of the network, and its main function is the packets scheduling according to the previous labels that the edge device set into the

packet header. With these two functions, websites implement their own QoS policies. The packets colouring is based on two colours: green and red. Packets that need to receive the highest priority are marked as green, therefore, the rest of the packets, those packets that do not need to receive so high priority, will be marked as red, and therefore, will receive a lower priority at the core device.

With the above specifications, and considering that the packets processing at the edge and the core devices is done at network level, the final size of flows is unknown, therefore, the only possible mechanism to implement is through a flow size threshold: when the size of a flow is lower than a certain threshold, it receives the highest priority level, and therefore, the rest of flows receive the lowest priority level because they are treated as long flows. In order to illustrate the scenario, if we have configured a threshold of 13 Kilobytes, with the above specification, flows smaller than 13 Kilobytes will receive always the highest priority treatment. On the other hand, flows higher than the threshold, will receive always, the lowest priority.

Our first question is: *how could we avoid a penalisation of flows whose size is close to the threshold?* With the traditional proposals, flows that are a little longer than the threshold would be sent to the lowest priority queue. For example, flow sizes of 13.5 Kilobytes will receive the lowest priority.

In this thesis, we are looking for increasing the adaptability of this threshold. For this reason, we have added the packets promotion term to the DiffServ framework. Measuring the total incoming traffic to the DiffServ system, we propose to promote some long flows, from a low to a high priority status. With the last promotion operation, we fall in our second question: *How many flows can we promote while keeping a suitable system performance?*. For this target, we have added the token bucket mechanism to model the packets promotion in the QoS system. Obviously, we need to regulate the packets promotion: if we promote too many packets from long flows, they will affect over short flows, and the overall performance will be degraded. On the other hand, if we do not promote packets or promote too few packets, the mechanism will be negligible.

Once we have developed our promotion mechanism, the third question raised: *are we promoting some flows that do not need to be promoted?*. To solve this question,

we propose another threshold to detect those extremely long flows.

And our last question is: *How can we make the extremely long flows threshold adaptive?*. In order to answer, we propose an adaptive threshold to differentiate extremely long flows that will be computed from an historic record of recent past flows the QoS system.

Finally, in order to compare the performance of our proposal with other solutions, we have selected other algorithms as Short Flow Differentiation (SFD), Random Early Detection (RED) and DropTail to run in the same architecture and with the same simulation conditions.

Chapter 2

Background and related work

The enhancing of the QoS of web traffic can be done considering multiple points of view but we have identified three main aspects that are depicted in the mind map of the Figure 2.1 in order to organise the analysis: *features*, *foundations* and *proposals*. The above three characteristics are interconnected with each other in such a way that certain well know features, such as, pattern traffic classification are directly related to some foundations, i.e., DiffServ architectures, and finally, close to classical flow differentiation.

2.1 Basics and foundations

In this section, we will briefly detail the foundations that support the QoS in general. We have analysed them from several points of view, starting by a brief summary of the QoS foundations and terminology. The DiffServ architecture and the token bucket model are also described in this section because they are widely developed in the QoS context and this thesis is based on these two concepts. Finally, we deal with the most import facts related to AQM in the QoS from the web traffic management perspective.

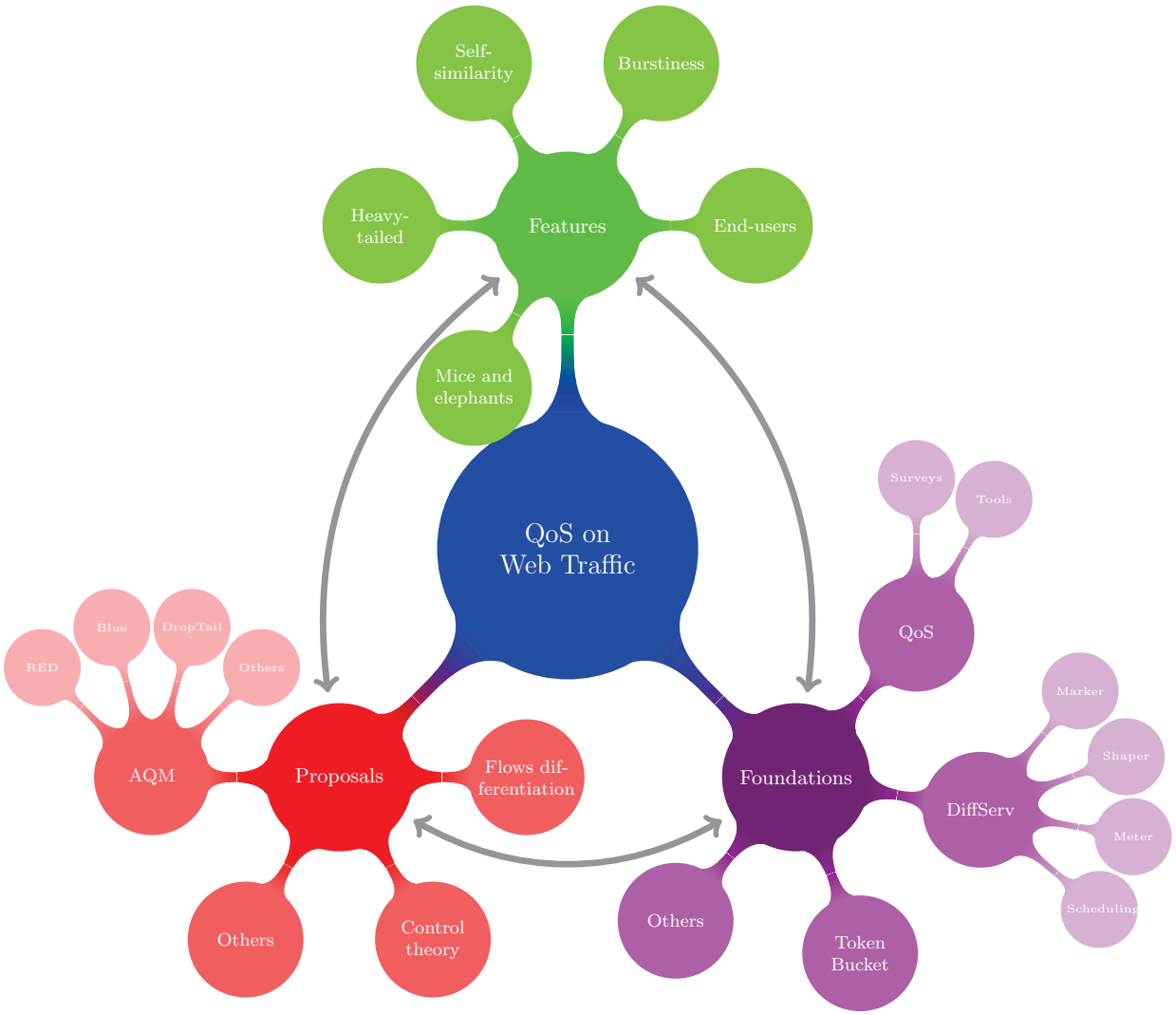


Figure 2.1: Different characteristics of the QoS.

2.1.1 Quality of Service

QoS engineering addresses the technical issues raised by the appearance of new types, classes and qualities of Internet services. Everyday, new services and more traffic demands are required to be supported by the Internet infrastructure. QoS issues have been widely analysed from the early days of Internet in multiple surveys and studies that explain the components, frameworks and characteristics of the solutions proposed by the researchers. In this section, we review the literature that has either a connection to QoS in general or some individual technique that is used in this dissertation.

The terms and frameworks for supporting QoS were developed on the early 90s. The first applications were oriented to get a framework that permits the multimedia data to achieve a certain transmission quality level. Some of those first papers that we can find are [96, 36, 55, 16, 39]. They introduced the first effort to apply to the Internet the novel technology recently developed in those years.

The first paper that details the QoS principles is [6], where the QoS terminology in the context of distributed multimedia systems is described. This is one of the most detailed exposition about the QoS architectures that have emerged in the literature. This work addresses the QoS from threes points of view: *principles*, *specification* and *mechanisms*.

Regarding the QoS *principles*, there are several issues identified in the QoS specification:

- The *transparency* principle indicates that applications should be shielded from the complexity of the underlying QoS specification and QoS management. The target of this principle is to reduce the embedded QoS functionality over the application level in such a way that the QoS functionality is developed in the internetworking devices such as routers and switches.

- The *integration* principle states that each layer of the Open System Interconnection (OSI) must collaborate to achieve the QoS requeriments. From the Application to the Physical layer, flows go down through the different resources

and protocols and each resource module must provide its own QoS characteristics.

- The *separation* principle establishes that data transfer, control and management are network tasks which are functionally separated, and these tasks should be separated in the QoS architecture framework.

- *Multiple time scales* principle states that there are different time scales for each process (e.g., routing protocols, scheduling, QoS management, etc.).

- The *performance* principle subsumes a variety of agreed rules for the implementation of QoS communication systems.

The QoS specification is concerned with QoS application-level and management policies. QoS specification is generally different for each QoS system and it is used to configure the QoS mechanisms located at the end-system and network. These QoS specification could be summarised as follows:

- *Flow performance specification*, related to the specification of different traffic parameters as: jitter, throughput, delay, packet loss, etc. Obviously, the selection of the most important traffic parameters depends on the QoS system.

- *Level of service*, which establishes the degree of the end-to-end resource commitment required.

- *QoS management policy*, which states the degree of QoS adaptation than flows can tolerate and the actions to be taken in case of not accomplishing the level of service.

- *Cost of service*, related to the price that users pay by this service.

- *Flow synchronisation* specification, when it is necessary the synchronisation between several flows.

Aurrecoechea *et al.* developed a wide QoS specification by proposing the QoS mechanisms that a QoS system must provide and include in their architecture. In

this document, QoS mechanisms are divided into three categories: *provision*, *control* and *management* mechanisms.

Regarding to the *QoS provision mechanisms*, they must provide the following functions:

- *QoS mapping*, which connects the QoS specifications between different levels depending on the required services, for example, at transport level to accomplish a certain parameter, such as delay, jitter, etc.
- *Admission testing* by controlling the resouces required by the incoming traffic to the QoS system.
- *Resource reservation* protocols, that are related to the resource provisioning and closely connected with mapping and admission testing.

Next, the *QoS control mechanisms* are presented, which are related to the devices and networking media and manage the time parameters. The QoS control mechanisms comprise the followings functions:

- *Flow scheduling* that deals with flow forwarding, queue management and scheduling policies at the appropriate network devices, such as core devices in the DiffServ environment.
- *Flow shaping*, which is used to regulate the incoming traffic to accomplish the QoS specification, for example, peak rate and burstiness.
- *Flow policing*, which is applied for monitoring tasks. The monitoring operation can be associated to the QoS management, controlling the agreement contract between provider and client.
- *Flow control*, which classifies the flow control in two categories: open-loop flow control for telephony networks and closed-loop flow control where the sender rates can be adjusted based on the receiver feedback.
- *Flow synchronisation*, which is related to the event ordering and multimedia interactions.

And finally, the *QoS management mechanisms*, that are used for supervision tasks ensuring that the QoS contracted is guaranteed. The following management mechanisms are proposed:

- QoS *monitoring* in different networking levels and time scales, scheduling policies, flow shaping and other control mechanisms that have been introduced above.
- QoS *availability*, which allows to specify the interval over some performance parameters are examined.
- QoS *degradation*, regarding to the impossibility to achieve the QoS contracted. It is often used in the client side providing other lower QoS levels.
- QoS *maintenance*, comparing the expected QoS performance with the QoS monitored.
- QoS *scalability*.

On the late 90s, we can also find papers that are more specifically related to switched packet networks. For example, Guerin *et al.* [42] reviewed the basic mechanisms used to support QoS features in packets networks, focusing on the analysis of the different scheduling and buffer management mechanisms. The study emphasised the need of adapting solutions for different environments where the QoS is deployed and finally, points out the idea that multiple mechanisms can coexist in the same network resource in order to guarantee the end-to-end QoS. From the first steps of the QoS technology, the different subjects involved have demanded different service requirements, structuring the Internet traffic over different traffic classes and therefore, demanding different levels of quality of service for each traffic class. Other interesting study of that decade was Xiao *et al.* [104], where they presented a big review of the emergent technologies in that time: Resource Reservation Protocol (RSVP), DiffServ, Multiprotocol Label Switching (MPLS). They discussed the available implementation of these technologies and the emerging problems.

The most important research contribution in the QoS development was produced in the early 2000. In particular, for switched packet networks, we can find multiple contributions such as Bauer *et al.* [9], Lu *et al.* [60] and Goldsmith *et al.* [41].

Other up-to-date study related to the QoS architecture was Zhou *et al.* [110]. In this paper it was stated a wide variety of approaches and metrics to deal with the problem and many QoS points of view are developed. In this sense, this work set up three categorisations to classify the whole QoS problem. The first category was related to the locations where QoS features are applied, that are the server, the proxy and the network side. Specifically, at the network side, the QoS differentiation was mainly based on the DiffServ architecture, that performed the flow differentiation in both the edge and core devices. The QoS differentiation approaches were also categorised regarding policies such as *admission control*, *resource management* and *content application*. And finally, the third category was based on the implementation level, such as the application and kernel level.

The Next Generation Networks (NGN) technology was launched by the International Telecommunication Union (ITU) and European Telecommunications Standards Institute (ETSI) as a method for establishing convergence of Internet Protocol (IP) communications. Although this new networking concept was mainly focused on the wireless technology, the traditional QoS concepts and methods continue being absolutely valid as we can observe in references [69, 95, 91, 73].

The mathematical models that support QoS methods are out of the scope of this thesis. For this reason, we recommend some references to complete the QoS review [35]. There are also multiple books and manuals related to QoS where the general concepts, trends and techniques are explained [34, 87, 102, 5, 101, 83].

2.1.2 DiffServ

DiffServ is a networking architecture which defines a simple and scalable mechanism for classifying and managing network traffic providing QoS. The first document about DiffServ was published in December 1998 by the Internet Engineering Task Force (IETF) [11], and it defines a framework that supports a scalable form of QoS.

DiffServ operates at class level, where a class is an aggregate of many such flows. For example, a class might include packets coming from a set of source addresses or packets of a certain size. DiffServ architecture is based on the basic Internet philosophy, where the complexity is relegated to the edge device while preserving simplicity of the core device. Per-hop behaviour (PHB) policies have been standardised in two classes by the IETF: *Expedited Forwarding (EF)* [53] and *Assured Forwarding (AF)* [47]. The main goal of the PHB-AF is to deliver the packets reliably. It is suitable for non-real time services such as Transport Control Protocol (TCP) applications. Other remarkable reference in this area is the RFC 3246 [27] which states features related to low-loss and low-latency traffic.

The main contribution of the DiffServ framework was the packet marking. This characteristic includes IP-layer marking at the edge device under a certain criteria. After packets are marked, the second task is scheduling at the core device, where packets are sent to different queues by following the appropriate scheduling policies. There is a wide literature about how packets are marked at the core device. A three-level colouring was defined by Mo [67], that achieves a fair bandwidth distribution mechanism in a DiffServ network. Yeom *et al.* [107] detailed a simple scheme for improving the service provided to a receiving-intensive application by transferring resources to the edge of the network on the sender's side, and they also studied the impact of this sender's side marking strategy and the receiver's willingness to pay for resources when achieving QoS goals of individual flows. Mellia *et al.* [65] considered an alternative strategy to RED with IN and OUT drop probability (RIO), in which packets are marked based on the state of individual TCP flows. Rossi *et al.* [82] offered a simulation study of Hypertext Transfer Protocol (HTTP) traffic over a DiffServ architecture. This study analysed the HTTP performance in different congestion scenarios, when the reserved bandwidth is able to support the offered traffic (*overprovisioned*) and, in the opposite case, when the reserved bandwidth is not able to support the HTTP offered traffic (*underprovisioned*). They concluded on their study that the DiffServ AF was able to achieve the compromised QoS in a overprovisioned context, while the compromised bandwidth was not guaranteed in a underprovisioned context when the fairness was divided among short and long flows.

These results illustrated the idea that it was important to establish a differentiation between short and long flows in order to avoid an unfair sharing of the available bandwidth.

After the DiffServ architecture was proposed and during the 2000 decade, different proposals appeared oriented to improve the QoS using this new architecture. One of the firsts was Xiang *et al.* [103]. They proposed a packet classification and dynamic queue management based on DiffServ and RED queue management. The simulation results show that this algorithm consumes less network resources than RED and it also improves the fairness among different types of flows. During this decade, some new network services appeared and the discussion was oriented towards introducing QoS in DiffServ. In this context, Tsolaku *et al.* [97] prepared a detailed study of the network services defined and deployed within the DiffServ architecture and they also proposed four more variations. They concluded that the traffic handling mechanisms were suitable for the proposed network traffic services.

Insisting on the idea of fairness regarding the bandwidth distribution between short and long flows, there are many references in the literature. Siris *et al.* [86] analysed the fairness consequence of the network sharing. Their contribution was implemented at the network edge, proposing a simple adaptive marker in this node. In this work, they proposed a bandwidth broker architecture where this element drives the control parameter inside the marker function based on the current traffic conditions. The concept of packets marking at the edge device and the unfairness problem produced in a DiffServ architecture were also covered by Elshaikh *et al.* in [31]. Where they proposed an improved version of the time sliding window three color marker. In this version, they included traffic adaptability mechanisms and improved the results obtained by previous algorithms such as srTCM [48], trTCM [49] and ItswTCM [90], in terms of fairness.

From the early days of the DiffServ development until nowdays, the DiffServ architecture continues being valid and used in most of the QoS systems at the Internet Service Provider (ISP), as we can confirm in the next references where new proposals based on packets coloring and the unfairness share problem are analysed in [92, 3, 59].

As it was mentioned in the last section, the mathematical model is out of the scope

of this thesis, however there are many studies related to the DiffServ and packets differentiation such as [70].

2.1.3 Token bucket

The token bucket model was defined in RFC 1363 [76] as a flow specification in Internet and mainly used in DiffServ. The token bucket traffic shaper is one of the most important traffic conditioners that we can find in QoS architectures and it is defined with two parameters: Committed Information Rate (CIR) and Committed Burst Size (CBS). The main function of the traffic shaper is to avoid the traffic peaks that overload devices and, therefore, introduce an excessively traffic delay in bursty traffic environments. Regarding the token bucket traffic conditioners, we can find multiple versions of the original model. For example, the proposal of Tang *et al.* [78], that adds a new queue to the original token bucket model to accommodate those packets that are not able to be delivered because there are not enough tokens at the bucket. This new queue has a smoothing effect over the incoming traffic and permits to accommodate certain non conforming traffic. Other researchers as Yang *et al.* [105] demonstrated that the QoS weakly guarantees the service quality when the token bucket profile produces a hard coupling between the average rate control and the burst size control. For this reason, their proposal decouples the long term average rate control from the burst size control. This solution improved the QoS level for conforming traffic compared to the traditional token bucket specification.

The token bucket marker and DiffServ architecture are closely related when it is needed to count the incoming traffic to accommodate the traffic to the contract conditions (Service Level Agreement (SLA)) or any other constraints. The token bucket model is usually set up at the edge devices, where incoming packets are marked. In this sense, Park *et al.* [75] proposed an adaptive token bucket algorithm to solve the unfairness share among aggregate flows in DiffServ networks, and they proposed to adjust the target establishing a feedback mechanism edge-to-edge and any other additional signalling protocol or measurement. Their simulation proposal provided fair sharing bandwidth over different scenarios and network conditions.

As it has been mentioned previously, the token bucket operation had been widely analysed in many studies considering the individual point of view of each device in the token bucket model. That is, the marking process at the edge and the scheduling task at the core device, and the effect over the specific token bucket parameters, such as CIR and CBS. Sue *et al.* [89] analysed the combined effect of both processes and proposed a model that connect the CIR, CBS, Round Trip Time (RTT) and the packet dropping at the core device. They validated the model through simulation and confirmed the fairness effect in the bandwidth share.

Recent applications of token bucket marker that improve the perceived customers quality of service can be found in Farmet *et al.* [33].

There are also several analytical models describing the token bucket model such as [2, 88].

2.1.4 AQM

Each application protocol in Internet generates a different traffic workload. Each type of traffic normally shares 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 [26]. For this reason, it is necessary to develop more sophisticated queue management that avoid this drawback. The most important AQM mechanism was proposed by Floyd *et al.* in [38]. They proposed the RED algorithm that focuses on detecting and avoiding congestion in networks and the global synchronisation problem which is very pernicious in switched packet networks. With RED queue management, packets are dropped with a certain probability before the queue reaches an overflow state.

Before the implementation of RED as a standard in the switched packet networks, DropTail was the only available mechanism to schedule packets in the switched networks. DropTail is mainly based on FIFO queue management and it is not able to accommodate different types of flows to achieve any QoS. The advantage of RED over DropTail was analysed by Brandauer *et al.* in [13]. This advantage is mainly

because RED does not permit an excessive queue length, especially during peak load and congestion conditions. This feature allows to accommodate bursts of packets into the available queue and, hence, to achieve an overall performance improvement of the shared network.

The original RED version has been widely used to improve the web traffic performance [12, 21]. In fact, several versions of the original RED algorithm have been proposed. For example, Claypool *et al.* [23] presented SHort-Lived flow friendly RED (SHRED), that uses an edge hint to indicate the congestion window size in each packet that is sent by the flow source or by an edge router. SHRED drops more often packets from long-lived flows than from short-lived flows. By contrast, Altman *et al.* [4] considered three variants of RED and compared them to a DropTail buffer: standard RED, adaptive RED [37] and a *gentle* option of RED. Wang *et al.* [100] proposed Subsidised RED (SRED) that targets short-lived or fragile flows to keep the link utilisation high, while reducing the average flow response time. Effective RED (ERED) was proposed by Abbasov *et al.* in [1], and aims to reduce packet loss rates in a simple and scalable manner. Long *et al.* [57] conducted an empirical study of the effects of AQM policies on the distribution of response times by comparing three schemes: Proportional Integrator, Random Exponential Marking (REM) and Adaptive RED.

2.2 Web traffic features

Since the World Wide Web (www) was developed by Tim Berners-Lee [10] working at CERN, in Geneva, the HTTP has been the communications protocol most widely used in Internet [77, 24, 99]. Different applications (kazaa, P2P, Ajax, Youtube, etc.) and features (web 2.0) have been recently added to the Internet traffic, nevertheless the latest studies show that web traffic is still the most usual data flow in Internet [51, 28, 79, 93]. At the early stages, 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 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 [50, 80]. Although HTTP does not provide any QoS, different web users share the available bandwidth and the network resources of the ISP. In this context, small web pages requested from clients coexist with video streaming and database queries.

Some of the most accepted parameters are related below: *flow size*, *duration*, *burstiness* and *rate*. These parameters have been used to make up different flow classifications comparing each type of flow with a different animal attending to their own characteristics. The flow classification is summarised in Table 2.1.

Parameter	**Description**	**Types**
Size	Amount of data transported	Mice, elephants
Duration	Time interval from the first packet until the last packet	Dragonfly, tortoise
Burstiness	Related to the packet inter-arrival time (lower than a defined threshold)	Alpha, Beta
Rate	The incoming data rate to the system	Cheetahs, snails

Table 2.1: Flow classification.

Related to web traffic, five characteristics have been identified: *mice and elephants phenomenon*, *heavy-tailed distribution*, *self-similarity behaviour*, *burstiness* and *end-users requirements*.

2.2.1 Mice and elephants paradigm

As we have illustrated in the Table 2.1, Internet flows are classified into several terms related to the analogies with certain animals. We only use those that are related to *size classification (mice and elephants)*.

Since the early 90s, the Internet pattern characterisation has been targeted by many researchers. For example, Claffy *et al.* [22] developed a parametrisable methodology for profiling Internet flows with several granularities. Thompson *et al.* [94] monitored and analysed the traffic over several high bandwidth backbones and the research concluded with an extensive and detailed analysis of the Internet traffic in

terms of packet sizes and duration. They also analysed the flow composition by several criteria about protocols and applications, for two time scales (24 hours and 7 days). In order to asses if the Internet pattern classification was scaled to every Internet levels, Fang *et al.* [32] reported an study of traffic patterns among autonomous systems. This work displayed a highly non-uniform distribution of flow traffic between a pairs of hosts, networks and autonomous systems. They presented some numerical pattern distributions and concluded that the top 9% of flows between autonomous systems accounts for the 86.7% of the packets and the 90.7% of the bytes transmitted. This work also suggested that routers need to maintain a limited QoS flow state.

Reviewing the literature, the *mice* and *elephants* terms were coined by Zhang *et al.* in [109]. The *mice and elephants paradigm* is well documented by the scientific community and it is described as follows: most of Internet flows carry a short amount of traffic, while the rest of flows represent most of the traffic. These types of flows are named *mice* and *elephants*, respectively. As the *mice/elephants* classification is based on the flow size, many researchers have driven their investigations in order to look for the bound which flows left the *mice* state and reach the *elephant* state.

The terms mice and elephants and the flow differentiation are closely linked because before proposing an specific solution for each type of flow, the first task is to differentiate and isolate each type of flow. The flow differentiation is usually based on DiffServ architectures and AQM solutions. Guo *et al.* described in [43] one of the first analysis about the interaction between *mice* and *elephants*, by proving that *mice* are defenceless against *elephant* flows, tending to loose the link bandwidth tipping the scales in elephants favour. For that reason, this research established that mice flows need a special treatment, proposing to use the RIO active queue management inside the DiffServ architecture to give preferential treatment to short connections in a bottleneck queue. They focused their analysis in heavy load scenarios and establish that short flows need to be protected against long flows in terms of fairness and response time. Their solution works as good as the RED scheme in terms of response time and goodput. Guo's proposal solved the short flows protection but this solution considers only a fix threshold value that might not be suitable for other traffic patterns.

When flows from different nature share the same media, the unfair distribution

of the media appears immediately, and long or persistent flows acquire more bandwidth than short or non-persistent flows. Classification mechanisms must consider this feature and propose solutions to alleviate the bandwidth sharing of short flows. Kantawala *et al.* proposed in [54] a solution for the problem of large delays that packets may suffer at the router devices, mainly backbone routers under heavy congestion traffic conditions. They proposed a more sophisticated packet schedulers that improve the performance results for short flows. For example, Chait *et al.* [19] proposed to classify and separate different flows into separated queues at the core routers. This solution provides better fairness and improves the predictability, the transmission delay and the better control over the QoS. The same reasoning is followed by Yilmaz *et al.* [108] who proposed a class-based isolation of flows using the flow size as a criteria to enforce the fairness when the congestion level is incipient. With this solution, they achieved the improvement of the following features: fairness, predictability apart from a lower transmission delay and better control of the QoS.

It has also been analysed the mice and elephant effect in Internet connections when analysing TCP protocol measurements. Several studies have taken into consideration that most of the TCP connections are made up of a few amount of packets. For example, Avrachenkov *et al.* [7] suggested the use of scheduling algorithms which favour short jobs, such as Least Attained Service (LAS), to differentiate between short and long TCP flows. They proposed a packet level stateless threshold based on a scheduling mechanism for TCP flows. This property was also considered by Rai *et al.* [52] using the LAS policy to improve the response time of web traffic. The advantages of scheduling algorithms to benefit short jobs were discussed in [44], where they proposed to perform differentiated control over Web-based transactions to give preferential service to short web requests. Similar to the aforementioned works, Harchol-Balter *et al.* [46] proposed to give preference to those requests which are short, or have small remaining processing requirements, in accordance with the Shortest Remaining Processing Time (SRPT) scheduling policy.

From the theoretical point of view, the analytical model for short-lived flows has been the focus of several studies. For example, Cardwell *et al.* [18] described the steady-state throughput of TCP flows with extension to the start-up connection

phase. Mellia *et al.* [64] proposed a recursive and analytical model to predict the TCP performance in terms of completion time for short lived flows. This proposal was based on the knowledge of the average dropping probability, the average RTT and the flow length.

Brownlee and Claffy [14] did not only consider the mice and elephant classification based on the flow size. They introduced the concept of network traffic streams and suggested the ways they aggregate them into flows. They proposed a method of measuring the Internet flows in terms of size and lifetime. According with the above characteristic, they define the dragonflies and tortoises flow types. Therefore, a *dragonfly* must be a flow with a short cycle-live and faster. This description includes flows that last less than 2 seconds, that according to their measurements, are about the 45 % of the total number of flows. Meanwhile, there are long-lived flows with lifetimes of hours to days that carry a huge amount of data (about the 50-60 % of the total number of bytes of a link). In this regard, they introduced a second term related to the flows, the *tortoise* flow, as the flow that are longer than 15 minutes. They underlined that flows can be classified from the traditional point of view, considering the parameter size, and from their point of view, considering the lifetime. They considered that flow sizes are independent of flow characteristics, however, both are important to understand the overall flows behaviour in Internet.

Papagiannaki *et al.* [74] emphasised the elephant flows effect over some network functions such as re-routing and load balancing, where some traffic engineering applications could exploit this phenomenon treating elephant flows in a differentiated manner. This work analysed flows bandwidth in order to detect elephant flows by collecting the overall data of the link and classifying the flows based on both volume and persistence in time. The main idea of the elephant classification is about the definition of a separation threshold that elephant flows have to exceed. The initial threshold value is identified through two different procedures:

- *aest*: based on the heavy tail nature in the flow bandwidth observed in the collected data

- α-*constant load*: this strategy needs an input parameter corresponding to the

fraction of total traffic that will detect the elephant class.

The conclusion of this work is that even this is a simple proposal and the classification scheme is able to detect elephant flows, it is insufficient for most traffic engineering applications.

Also considering the elephants detection, Mori *et al.* [68] proposed that if we are able to obtain an statistical approach of this type of flows, it will be very useful for network operation and management. Using packet sampling, they proposed a technique based on the Bayes' Theorem to identify elephant flows. Deb *et al.* [29] emphasised the fact that, even with enough bandwidth available in networks, it is necessary a flow differentiation to avoid a poor QoS. This is the motivation for differentiated services, specially for web traffic.

Gandouet *et al.* disserted in [40] about the problem of estimating the number of elephant flows in a IP stream. Exploring some theoretical space complexity of this problem, they concluded that it cannot be solved with a complexity less than $O(n)$. They proposed the LOGLOG algorithm which returns an estimator of the number of elephants while using a small amount of memory.

As it has been summarised in Table 2.1, several researchers have classified the Internet flows using different parameters. Lan *et al.* have studied in [56] the correlation between each type of flow attending the parameters: size, duration, rate and burstiness. They have found a certain correlation between some combinations of size, rate and burstiness and they justified this correlation by the application and transport-level protocols.

Recent Internet traffic studies, for example, Quan *et al.* [81] assessed and proved that the classical mice and elephant phenomenon is valid nowadays. In this work, the relation between mice and elephants is confirmed, finding that about 20 % of the overall traffic is carried by long flows.

2.2.2 Heavy-tailed distributions

At the same time as researchers were focusing on the Internet traffic classification, they were discovering evidences of the heavy-tailed pattern of the file sizes that are

transmitted in the web traffic. One of the first studies was done by Crovella *et al.* in [25], where the heavy-tailed evidence was underlined from different measurements: file requests by users, file transmissions through the network, file transfers time, and files saved on servers. Zhu *et al.* proposed in [111] a model to produce statistics for file transfers according to the heavy tailed traffic distribution in Internet. By using this model, they provided a natural and plausible explanation for the origin of the heavy tail distribution in Internet, and they concluded that this feature is a permanent and ubiquitous characteristic of Internet traffic and it does depend on each protocol or user behaviour. More recent studies confirmed the heavy tailed characteristics of the web traffic such as, Miller *et al.* [66] that analysed several logs from different web servers and different applications (i.e. markets, bank, educational institutions) from the point of view of the session workload. They checked that the data mining is also dependent of session workload and they have also concluded that session workload, follows a heavy-tailed distribution for transmitted data.

2.2.3 Self-similar behaviour

The fractal theory [62] was introduced to web traffic by Marie *et al.* in [63]. At the beginning of the Internet, some authors detected self-similar characteristics like the fractal's theory in web traffic. Therefore, Crovella *et al.* [26] applied the notion of self-similarity to explain the behaviour of web traffic analysing several high density traffic backbones in USA. They evidenced that the origin of the self-similar nature could be found in the document sizes distribution of web pages, the caching process and the user preferences. Recent studies, for example, Shiyin *et al.* [85] showed that the fractal theory is still current to explain TCP characteristics such as the bandwidth and other issues related to the end to end congestion control.

2.2.4 End-users perception

The subject related to *End-users QoS expectations* has been widely discussed in [106, 72, 45, 58, 15]. It is an important key 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, end-users expectations should be strongly considered during the website development, and it is recommended to implement a suitable mechanism to improve the end-users perception about latencies and delays.

Chapter 3

Long flow promotion

Traditional QoS strategies are mainly based on marking flows for its differentiation over DiffServ. This architecture is based on two dedicated internetworking devices: the *edge* and the *core*, as it can be observed in Figure 3.1. Web requests are produced in the web client cloud, then they go through the Diffserv area and finally they reach the web server cloud. Hence, web responses follow the opposite way, from web servers to web clients. The edge node marks packets by adding different labels to them. The information contained in these labels specifies the workload conditions related to the SLA contracted by the client. When the marked packets reach the *core* node, they are forwarded over different queues by applying the suitable AQM or an stochastic treatment in order to achieve the required QoS level.

Our proposal considers the work of Chen *et al.* [20] as a startpoint. They developed the *Short Flow Differentiation* (SFD) algorithm to reduce the user-perceived web latency by using a basic DiffServ framework for flow differentiation that is executed in the edge device. Incoming packets are then marked according to their own rules that are based on a fixed threshold of flow size. After flow differentiation, packets reach the core device and they are scheduled under priority scheduling. The core physical queue is managed under RIO active queue management. The RIO scheme has two virtual queues, named queue-IN and queue-OUT. Short packets are forwarded over the highest priority queue (queue-IN) and the rest of packets, which are classified as long flows, are forwarded over the lowest priority queue (queue-OUT).

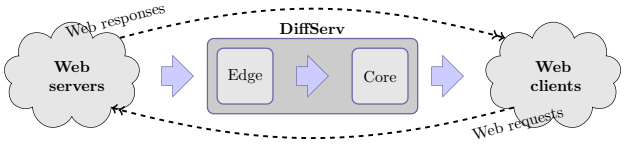


Figure 3.1: The DiffServ model.

SFD is also based on the mice and elephants paradigm. It assumes that the majority of flows in Internet are short because of the small objects from web pages. Although HTTP 1.1 allows persistent connections with more than one response on the same HTTP connection, most of the flows are short and only a few flows are long. Chen proposes a hard and static threshold based on the flow size. This threshold is obtained from diverse studies [20, 28] which state that the range `[13,15]` KB is accepted as the borderline size between short and long flows. Finally, they fixed the threshold in 13 Kbytes.

Obviously, with this solution, short flows always receive the most preferential treatment and, hence, achieve the lowest delay because they are forwarded over the most preferential queue. This high priority for short flows has an immediate negative consequence because long flows suffer a penalisation that in the majority of cases is an unnecessary penalisation. If the threshold is established in k packets, the first k packets of each flow receive the preferential treatment, but the following packets, $k+1, k+2, \cdots$ packets always receive a non-preferential treatment as they are forwarded over the low priority queue. This produces an overall increase of the long flow delay.

The second identified disadvantage in Chen's proposal is the static value for the threshold that does not take into account any traffic or environment information. This proposal does not adjust the threshold to any dynamic parameter. This fact can drive to a non appropriate packet classification at certain congestion levels. For example, considering a fixed threshold of τ packets, every flow of $\tau + 1$ packets will be marked as long flows and will be forwarded over queue-OUT at the core device. As the first k packets of each flow are marked as preferential packets, they are always forwarded over the queue-IN, therefore, they receive the highest treatment at the

core device. Obviously the remaining of packets are scheduled over the queue-OUT, receiving the lowest treatment. This fact means that from the $k + 1$ packet, the rest will suffer an unnecessary delay at the core queue and do not help in alleviating the overall delay.

This problem lead us to try to solve or minimise the effects over the overall delay and other performance parameters, such as jitter, dropped packets and overhead. We describe in this dissertation the development of a framework that is also based on the DiffServ architecture. We propose to add new characteristics to both devices. Firstly, it has been added a new differentiation algorithm in the edge device, and secondly, the core device has been modified by adding a new queue in this device. We also keep the queue management in the priority scheduling algorithm which is widely used as a device scheduler in a QoS environment.

The name of our proposal is LFP and it has been developed in a incremental way of three stages. The first state is based on the SFD mechanism, and introduces a prioritisation process of short flows based on the static threshold defined above. The second stage improves the first proposal by introducing the penalisation concept over the long flows. And finally, the third stage introduces an adaptive approach that detects and decreases the priority of really long flows.

1. Stage 1: Short flows over long flows (S_1)
2. Stage 2: Promotion of long flows (S_2)
3. Stage 3: Adaptive detecting and isolating elephant flows (S_3)

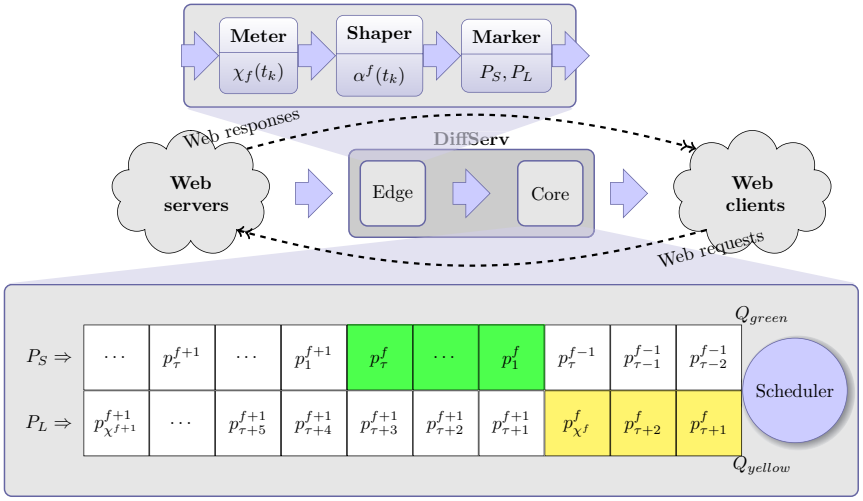


Figure 3.2: Stage 1: Short flows over long flows (S_1)

3.1 Stage 1: preferential treatment for short flows (S_1)

Considering only the HTTP responses from the Web system, traffic sent from Web servers reach the DiffServ area where it is firstly measured and classified at the *edge* device, as it is illustrated at the top of Figure 3.2. In this device, after going through the *meter* and the *shaper* processes, the incoming packets are marked in the *marker* process with different labels. When packets leave the *edge* device and reach the *core* device, they are sent over one of the queues depending on the assigned label. Finally, the Priority queuing (PQ) scheduling strategy at the *core* device configures the QoS level in the system [61, 30].

The overall incoming traffic is divided into n flows and defined as $f_1, f_2, ..., f_n$. Each flow f_i is composed of a sequence of p packets defined as: $p_1^f, p_2^f, \cdots, p_q^f$, where p_i^f defines the packet i from the flow f, as it is depicted in Figure 3.3. Let us define $\hat{f} = \{p_i^f, \forall i \in \{1, .., q\}\}$ as the set of packets from flow f.

The global amount of packets that arrive to the system are also numbered, independently of the flow they belong to. Hence, we consider that k^{th} packet arrives at instant t_k (seconds).

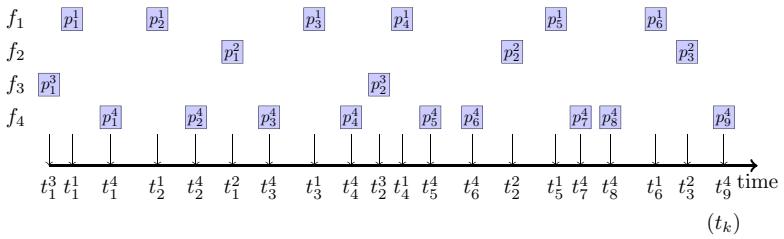


Figure 3.3: Example of a packet sequence in the QoS system.

Let us now describe the functions at the *edge* device. The *meter* is the first function, named as $\chi^f(t_k)$ that represents the number of packets of the flow f that have arrived during the interval $[0, t_k]$.

$$\chi^f(t_k) = \{Ord(\hat{f}) \text{ IN } [0, t_k]\} \tag{3.1}$$

After the *meter* process, packets go through the *shaper* function, that defines the transition from short to long state according to the threshold τ. First of all, the differentiation condition, $S_1(t_k)$, is established:

$$S_1(t_k) \equiv \chi^f(t_k) \leq \tau \tag{3.2}$$

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:

$$\alpha^f(t_k) = I(S_1(t_k)) \tag{3.3}$$

Where I function is the true function of S, defined as follows:

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

A set of labels is defined as $L_1 = \{P_S, P_L\}$ in order to mark packets in the *marker* process as belonging to short flows, P_S, or as belonging to long flows, P_L:

$$L_1 = \begin{cases} P_S & if\ \alpha^f(t_k) = 1 \\ P_L & if\ \alpha^f(t_k) = 0 \end{cases} \tag{3.4}$$

The core device is composed of the set of queues $Q_1 = \{Q_{green}, Q_{yellow}\}$, where Q_{green} is the highest priority queue and Q_{yellow} is the lowest priority queue. Labelled packets arrive to the *core* device and they are forwarded over an specific queue according to the label set inside the header of the packet:

- Packets with P_S label are forwarded over Q_{green}
- Packets with P_L label are forwarded over Q_{yellow}

Therefore, the packet range $[1, \tau]$ of every flow is forwarded over the Q_{green} queue. For each flow f, the highest QoS level is assured for the sequence $p^f_1, p^f_2, \ldots, p^f_{\tau-1}, p^f_\tau$, whereas the packet range $[\tau + 1, \chi^f(t_k)]$ – that is, until instant t_k –, is forwarded over Q_{yellow}. Consequently, the sequence $p^f_{\tau+1}, p^f_{\tau+2}, \ldots, p^f_{\chi(t_k-1)}, p^f_{\chi(t_k)}$ is penalised with the lowest QoS level as it is illustrated at the bottom of Figure 3.2.

This stage introduces a hard threshold that clearly differentiates between long and short flows. The main drawback of this proposal is that flows whose size is greater but close to τ packets, are treated as *long* and, hence, queued to the non-priority queue that may lead to a long response time for the client. This problem is reduced in the second stage of LFP by promoting packets from flows considered as long to the priority queue.

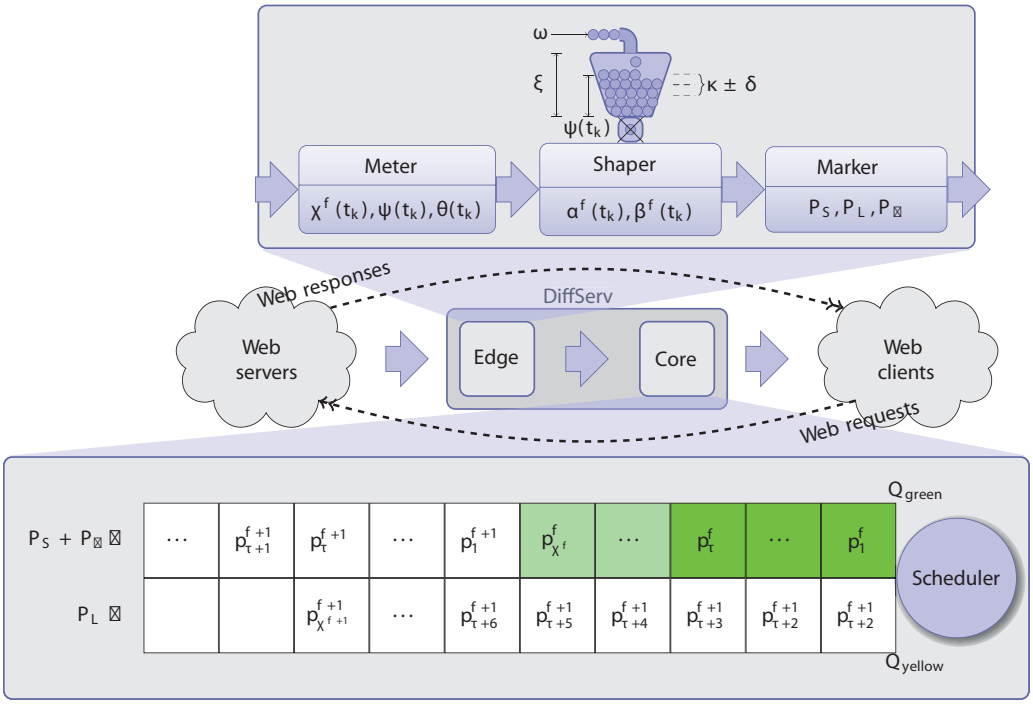


Figure 3.4: Stage 2: Promotion of long flows (S_2).

3.2 Stage 2: promotion of long flows (S_2)

In order to introduce more flexibility in the static threshold just defined, this second stage introduces a packet promotion mechanism to those flows that are a bit longer than the threshold (τ). The aim of this improvement is that packets from these flows are considered as they were belonging to short flows and, hence, receive a high priority QoS level when the system is not congested.

Previous stage (S_1) leaves some open questions: with $S1$, the first τ packets of each flow are treated with the highest priority and the rest of packets with low priority. Under certain traffic conditions and low congestion level, flows a little bit longer than the threshold could be treated in a different way by not being relegated to the lowest priority queue. In this sense, we could consider the following questions:

What does happen with flows a little bit longer than the threshold?

Why would we not to consider these special flows as short and treat them with high priority such as short flows in Stage $S1$?

In this thesis, the term promotion has been used for those packets that in S_1 would be forwarded over the lowest priority queue but under certain conditions they could be forwarded over the highest priority queue. The S_2 stages tries to promote these packets in order to enhance the overall performance system and, in particular, the overall flow delay. The implicit risk of packet promotion is to promote too many packets and not to achieve the expected performance results. For this reason, it is necessary to bound the promotion of packets by detecting the idle state of the network and identifying those candidate flows to be promoted. For this target, the token bucket model has been added to S_2. As in the stage S_1, the main philosophy continues being packets differentiation at the edge and packets forwarding over different queues at the core. The new functions have been added to the meter, shaper processes and we have modified the set of labels used by the marker function. Therefore, the token bucket model is used for two purposes:

1. Detecting the idle system state and allowing packet promotion from the low priority to the high priority queue.

2. Bounding packet promotion in order to prevent the congestion increase.

The token bucket function is described as follows: *the token bucket models the amount of packets that could be promoted from the lowest to the highest priority queue. The bucket is filled with tokens with a certain ratio depending on packets promotion and tokens that are spent when packets are promoted. Therefore, if there are tokens available in the bucket, packets could be promoted and tokens will be consumed and therefore, the token bucket will be filled out. On the other hand, if there is an incipient network congestion, no tokens are left for any promotion.*

In the next paragraph, we are going to model the token bucket and the mechanisms to fill up and empty the bucket. We define the maximum capacity of the token bucket as ζ (tokens) and the fill rate as ω (tokens/s). Considering $V^*(t_k)$ as the amount of

promoted packets during the interval $[0, t_k]$, the state of the token bucket at instant t_k is defined by $\psi(t_k)$:

$$\psi(t_k) = \psi(t_{k-1}) + IN(t_k) - OUT(t_k) \tag{3.5}$$

where:

$$IN(t_k) = min(\omega * (t_k - t_{k-1}), \zeta - \psi(t_{k-1})) \tag{3.6}$$
$$OUT(t_k) = V^*(t_k) * I\{V^*(t_k) \leq (\psi(t_{k-1}) + IN(t_k))\}$$

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

$$\theta(t_k) = \frac{\psi(t_k)}{\zeta} \tag{3.7}$$

However, we define some levels in the bucket in order to adjust the promotion. In order to control the promotion, it is desirable that the token bucket state, $\theta(t_k)$, remains close to a precise level or set point defined by κ, as it is depicted in Figure 3.4. As web traffic can presents peaks of traffic because of its bursty nature, $\theta(t_k)$ must range around κ with top and bottom bounds defined by δ. Therefore, the desirable working point for $\theta(t_k)$ is defined as:

$$\theta(t_k) \in [\kappa - \delta, \kappa + \delta], \kappa, \delta \in [0, 100], \kappa - \delta \geq 0, \kappa + \delta \leq 100$$

Parameters $\theta(t_k)$, κ and δ define the *open/close* state of the token bucket, and hence, the packet promotion. For this reason, to show the effect of κ and δ values over the token bucket state ($\theta(t_k)$), we consider that:

- If $\theta(t_k)$ is close to κ, the packet promotion is active.
- When $\theta(t_k)$ reaches $\kappa - \delta$ level, the token bucket will change to the *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 up with new tokens at ω ratio, then the token bucket level $\theta(t_k)$ will reach the $\kappa + \delta$ level again.
- In this point, the token bucket state will turn to the *open* state again and the promotion process will be reactivated.

Considering the above bounds, the differentiation function $S_2(t_k)$ can be defined as follows:

$$S_2(t_k) \equiv \big\{\theta(t_k) \geq \kappa + \delta\big\} \vee \big\{(\kappa - \delta \leq \theta(t_k) \leq \kappa + \delta) \wedge \beta^f(t_{k-1})\big\} \tag{3.8}$$

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

$$\beta^f(t_k) = I(S_2(t_k)) \tag{3.9}$$

The set of packet labels is now defined as $L_2 = \{P_S, P_L, P_\uparrow\}$, and it is used by the *marker* to accomplish the specifications according to the following rules:

$$L_2 = \begin{cases} P_S & \alpha^f(t_k) \\ P_L & \overline{\alpha^f(t_k)} \wedge \overline{\beta^f(tk)} \\ P_\uparrow & \overline{\alpha^f(t_k)} \wedge \beta^f(t_k) \end{cases} \tag{3.10}$$

Packets are then forwarded over the *core* queues as follows:

- P_S and $P_\uparrow$ packets over Q_{green}
- P_L packets over Q_{yellow}

With the above specification, the highest QoS level is guaranteed for the packet sequence $[1, \tau]$ of every flow, as they are forwarded over Q_{green}. Depending on the state of the token bucket and the incoming traffic, some packets from the sequence $[\tau + 1, \chi^f(t_k)]$ of some flows will be promoted over Q_{green}, whereas the remainder of packets, will be forwarded over Q_{yellow}. An example of this behaviour is depicted at the bottom part of Figure 3.4.

Packet promotion permits adapting the hard threshold of the previous stage of the algorithm to the workload. However, this approach includes a drawback under some circumstances of low congestion: when there are few flows going through 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 already expect a long delay for these flows. This can be considered as a waste of the available bandwidth, and hence, could produce an overall decrease in the performance of short flows. We consider this negative effect in the next stage and modify the approach accordingly in order to detect the extremely long flows.

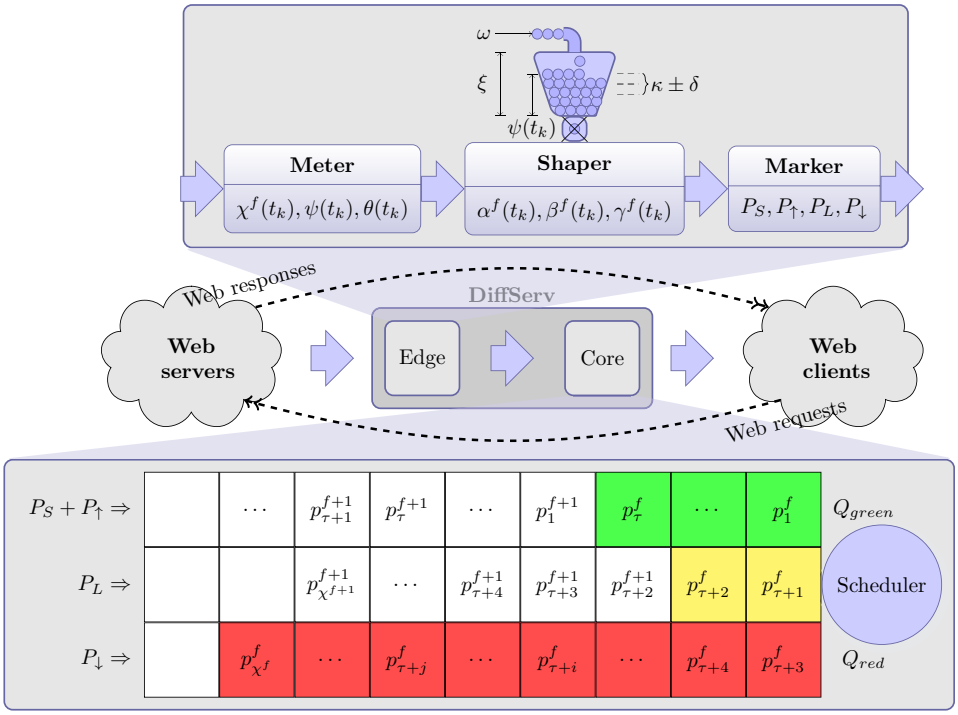


Figure 3.5: Stage 3: elephants penalisation (S_3)

3.3 Stage 3: detecting and isolating elephant flows (S_3)

As we have already introduced above, extremely long flows are normally named as elephant flows, and their presence in the priority queue must be avoided. Hence, in this stage our aim is to detect and isolate these flows in a new least priority queue.

Instead of defining a static threshold to detect these flows, we consider that the value that determines if a flow is an elephant flow has to also be adaptive to the traffic conditions of the network.

We have added new functions to the meter and shaper modules and modified the classification method by adding a new label in the edge device. It has been added a new queue for the scheduling packets at the core device. The global scheme is illustrated in the Figure 3.5.

The adaptive behaviour of this stage is based on an historic of the H last flow sizes

(in number of packets) in order to define the elephant threshold.

The presence of *elephant* flows in the QoS system is always bad news, as it produces an overall system performance decrease. 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 *elephants* and 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, setting a number of packets to differentiate flows as long or very long with a static threshold is not suitable, because an excessive low value could generate too many promoted packets and, by contrast, an excessive high value 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. Hence, we consider that it has to be calculated from the sizes of the last flows that have crossed the QoS system. Therefore, we consider $X_{t_k,H}$ as the set of the last H flow sizes that have gone through the system, and define it as follows:

$$X_{t_k,H} = \{\chi^f(i) \mid i \in [t_{k-H}, t_k], H \in \mathbb{N}, \forall f \in V(t_k)\} \tag{3.11}$$

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

$$Q_{X_{t_k,H}}(u) = Inf\{x \mid F_{X_{t_k,H}}(x) \geq u\} \tag{3.12}$$

As $Q_{X_{t_k,H}}(u)$ is calculated over a set of flows that have just crossed the DiffServ system, this measure provides an adaptive flow size measurement of the recent history of the flows that are crossing the system. As the S_3 goal is the detection and isolation of flows whose sizes are extremely long then the S_3 function is computed from (3.12) as follows:

$$S_3(t_k) \equiv \chi^f(t_k) \geq Q_{X_{t_k,H}}(u) \tag{3.13}$$

With the above definitions, $S_3(t_k)$ is the differentiating condition between *elephants* flows and just long flows. Hence, flows longer than $Q_{X_{t_k,H}}(u)$ are considered as *ele-*

phants, and should be sent to the lower priority queue. Otherwise, they are considered as flows with medium priority level. By applying (3.1) to the differentiation function, the function $\gamma^f(t_k)$ is added to the *shaper* module:

$$\gamma^f(t_k) = I(S_3) \tag{3.14}$$

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 incoming packets according to the following rules:

$$L_3 = \begin{cases} P_S & \alpha^f(t_k) \\ P_L & \overline{\alpha^f(t_k)} \wedge \overline{\beta^f(tk)} \wedge \overline{\gamma^f(tk)} \\ P_\uparrow & \overline{\alpha^f(t_k)} \wedge \beta^f(t_k) \wedge \overline{\gamma^f(tk)} \\ P_\downarrow & \overline{\alpha^f(t_k)} \wedge \overline{\beta^f(t_k)} \wedge \gamma^f(t_k) \end{cases} \tag{3.15}$$

When packets have left the *edge* device, they reach the next device at the DiffServ architecture, that is the *core* device (see Figure 3.5). This device is defined with the set of queues $Q = \{Q_{green}, Q_{yellow}, Q_{red}\}$. Packets marked as P_S or $P_\uparrow$ are forwarded over Q_{green} (highest priority queue); packets marked with P_L are forwarded over Q_{yellow} (intermediate priority queue) and finally, packets marked with $P_\downarrow$ are forwarded over Q_{red} (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.

As we have described above, the LFP algorithm has been described step-by-step and it is composed by three stages: S_1, S_2 and S_3. The final version of the algorithm includes all the stages and is named LFP for short. It has been summarised in the flowchart of the Figure 3.6.

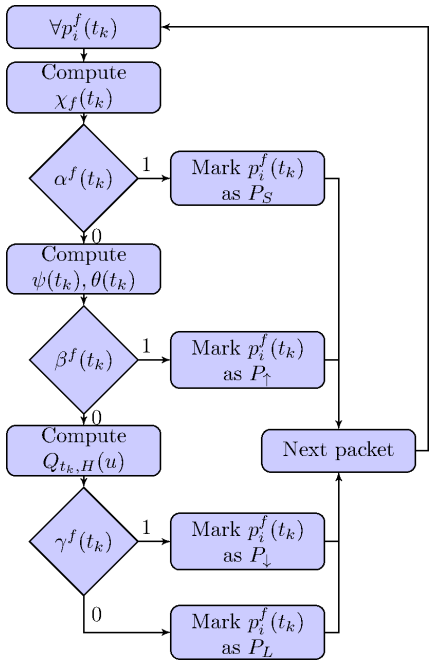


Figure 3.6: Flowchart for LFP with $S_1 \oplus S_2 \oplus S_3$.

Chapter 4

Performance results

4.1 Introduction

The performance results have been obtained through simulation using the network simulator ns-2 [71]. The considered scenario is based on a single bottleneck dumbbell topology where our DiffServ model has been implemented, configuring a link of 2 Mbps as the bottleneck with small buffers. The last analysis over Internet traffic suggest that large device buffers could be replaced with smaller ones [84, 98, 8]. Web traffic has been generated by the ns-2 extension named HTTP PackMime [17].

Based on the topology described above, different algorithms have been simulated over the same network and traffic conditions. Through simulation we have collected some results and analysed different performance parameters related the web traffic QoS. The results have been analysed from the point of view of the end-to-end performance and not taking into account the intermediate measures such as queueing delay or any other performance parameters related to the DiffServ system. The performance parameters analysed in this work are: latency, jitter, dropped packets, throughput and goodput.

The latency may be the most critical performance parameter in web traffic. It measures the time interval between the departure of the first packet from the server device until the arrival of the last packet of the message to the client device. As it has been mentioned previously, we have only analysed the web responses, from the server

to client. The size of these messages has a very high variability, from few bytes until mega or even gigabytes, while the web requests size is usually only few bytes. As the variability of web traffic from the point of view of flow size responses is very high, this type of flows suffers latencies that can vary from one second to many seconds or even some minutes. For this reason, service providers need to add new features to their QoS systems to improve clients satisfaction and browsing speed through their websites.

The jitter is related to latency or response time variation. This parameter is usually used as the performance metric of streaming protocols such as video or voice transmission. In our case, the jitter is used to evaluate the system capacity to keep the same latency over every packet of the flow.

The number of dropped packets is an obvious performance parameter. Dropped packets are always a negative circumstance in the switched packet networks. When there is a high number of dropped packets in the system, the rest of parameters are inaccurate, such as the final flow latency, throughput, goodput and, of course, traffic overhead.

The throughput and goodput are widely used as performance parameters in computer networks and traffic analysis. While throughput measures the amount of bits that are transported end-to-end, including headers, goodput measures only the application level data.

The overhead is the amount of extra data added to a message, which basically is the amount of bytes of the headers. This parameter provides the bandwidth consumed to send the message from a sender to a receiver.

Finally, we have also shown the link utilisation as the relation between the occupied data over the maximum link data ratio. This parameter shows the occupation level produced by each algorithm.

4.2 Parameters tuning

In this section we are going to detail the parameters used to configure the topology and algorithms. As we have mentioned above, the QoS system has been modelled in

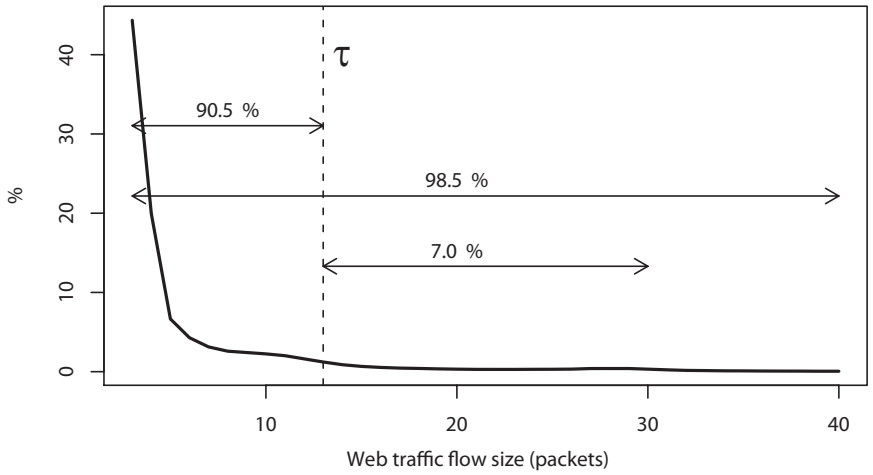


Figure 4.1: Flow size distribution for $\tau = 13\,packets$.

NS2. In this tool, both the *edge* and the *core* devices at the DiffServ architecture have been configured for RED management with the original values ($max_p = 0.02$, $w_q = 0.001$). The buffers capacity have been configured with a physical length of 50 packets with the RED parameters $min_{th} = 10$ and $max_{th} = 40$.

Regarding the algorithm configuration, the first task is to determine the threshold to differentiate between short and long flows. As it has been mentioned in previous section, the threshold is the amount of kilobytes which indicates when a flow changes its state from short to long. The value of the threshold (τ) that differentiates between Short flows (SF) and Long flows (LF) in web traffic has been analysed by Chen et al. [20]. They propose a table with five representative types of Web pages. The web page size average varies in the range of $[9, 12]\,KB$, from a minimum limit in the range of $[1, 3]\ KB$, until maximum values in the range of $[80, 90]\ KB$. Considering that the average web size has increased since then, we define an average flow size of 13 packets for our study, that would mean to define an average web page of 19 KB in an Ethernet infrastructure. The threshold τ has been approximated into packets for representative purposes.

As it has been mentioned in the background and related works, the heavy tailed nature of web traffic has been well analysed in many studies. The selected threshold and the simulated traffic used in this thesis recreates the heavy-tailed nature and the

mice and elephants classification. The Figure 4.1 shows the packet distribution for a certain congestion level and fixing the flow size threshold to differentiate between short and long flows in $\tau = 13\,packets$. As it is depicted in the figure, flows from the minimum size until 40 packets represent the 98.5% of the total workload in the system, which enforces the heavy-tailed nature of the simulated web traffic. Most traffic falls on the left side of τ, around the 90.5% of the overall traffic which corresponds to short flows and around the 10% falls in the range of $[\tau + 1, 40]$ packets, which corresponds to long flows. In the same figure, we can appreciate the existence of very long flows, named as *elephant flows* that are composed of more than 40 packets, and correspond to the 2% of the overall traffic.

Once we have checked the consistency of the simulated traffic generated with HTTP PackMime and the heavy-tailed characteristics of web traffic, we need to assess that flow differentiation that we have established by fixing a threshold to differentiate between short and long flows is kept over the different traffic scenarios. As we have mentioned above, HTTP PackMime is the web traffic generator, which models the web traffic as a stochastic model based on real aggregated traffic collected on a backbone or high-speed access links. The synthetic Web traffic is expressed as a collection of independent TCP connections, characterised by several variables such as the arrival time of the connection, the round-trip time for the client and server, the number of request/response exchanges and sizes of individual responses and requests. The Web traffic intensity is modulated with the R parameter, that sets up the incoming traffic as the number of new connections per second introduced in the system.

Congestion	R	P	F
Low	6	93.96	32.73
Medium	10	162.29	52.84
Heavy	14	229.51	73.68

Table 4.1: Scenarios for simulation with different congestion levels.

In order to check the consistency of the flow size distribution among the different traffic levels, we have determined several congestion scenarios as we can see in the Table 4.1. Varying the R parameters, and through simulation, we have measured

other parameters in order to reinforce the congestion level for each scenario. The description of these new parameters are:

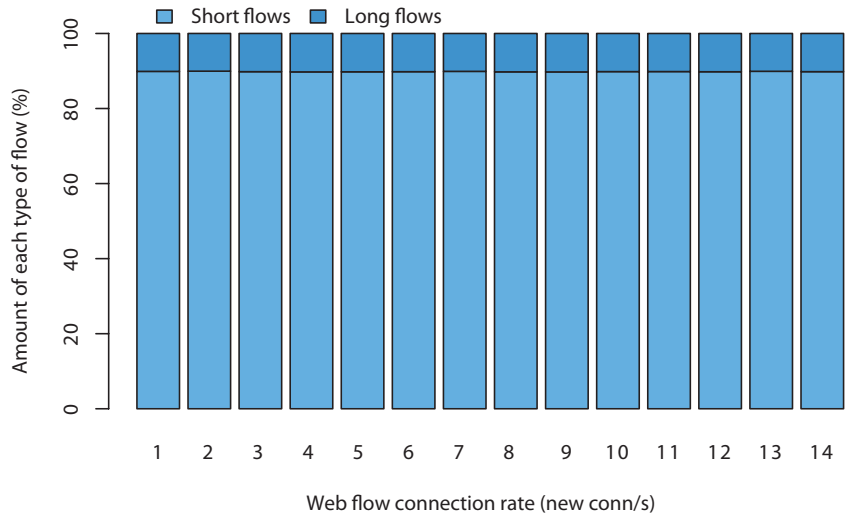
- P measures the new packets incoming to the system per time, and
- F, measures the new flows incoming to the system per time

The scenarios that we have configured are described below:

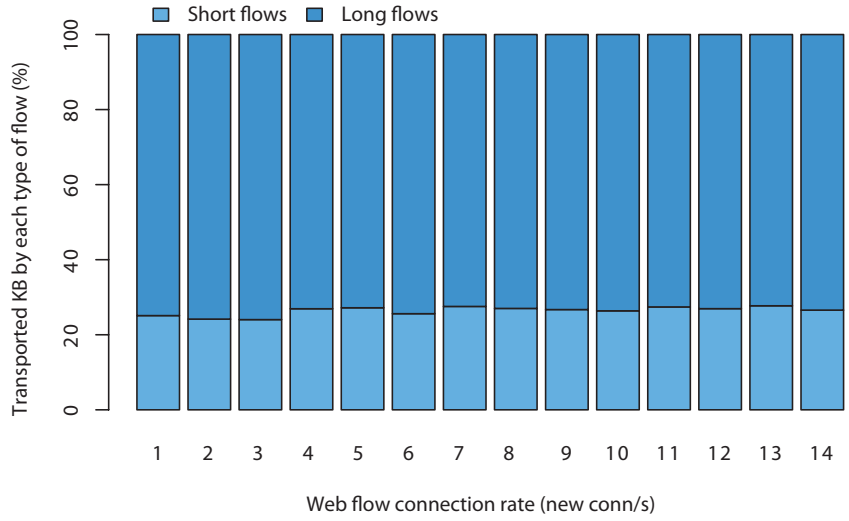
- *Low congestion:* in this scenario we set $R = 6$. The traffic intensity is very light and only a few packets are dropped due to the peaks of the incoming traffic.
- *Medium congestion:* this scenario has been configured with $R = 10$. The injected traffic to the system is close to the available bandwidth, therefore, some more packets are dropped and the congestion level becomes to be incipient.
- *Heavy congestion:* with $R = 14$ the incoming traffic to the QoS system is very high and the congestion level is severe. There are many dropped packets.

The three scenarios detailed above for different load in the system have been defined to evaluate the performance of our proposals over these scenarios. For this reason, we need to assess that the flow differentiation we have established by fixing a threshold to differentiate between short and long flows, is kept over every defined scenario. Therefore, we have obtained the simulation results from scenarios with very light congestion level ($R = 1$) until very hard congestion levels ($R = 14$). As we can observe in the Figure 4.2, the mice and elephants paradigm is accomplished for every scenario. The upper part of Figure 4.2(a) shows the ratio for short and long flows for each congestion level. Most of the traffic is composed of short flows up to the 80%, and therefore, the rest of flows are long. The bottom of Figure 4.2(b) shows the amount of transported information for each type of flow. As it is depicted, most traffic ($\approx 80\%$) is carried out but long flows, meanwhile the rest ($\approx 20\%$) is transported by short flows.

Another relevant element of our model is the token bucket mechanism. As we will verify after our simulation, the variation of the token bucket parameters have a low



(a) Flow distribution



(b) Packets distribution

Figure 4.2: The *mice and elephant paradigm.*

impact on the final performance results. They have therefore been set to $\omega = 1000$, $\xi = 500$, $\kappa = 40$ and $\delta = 20$.

We can consider several operating zones for the token bucket depending on the value of $\theta(t_k)$, as it can be observed in Table 4.2. The optimal operating point is located in Z_2, where $\theta(t_k)$ fluctuates in the range $[\kappa-\delta, \kappa+\delta]$. At this point, the token bucket state remains as *open*, and the packet promotion is produced according to the rules described in LFP Stage S_2. Meanwhile, Z_1 defines the *overrun* zone, where the normalised token bucket capacity $\theta(t_k)$ is below the desired operating point because the packet promotion rate is higher than ω. And finally, Z_3 defines the *underrun* zone, where the packet promotion is so low that the normalised token bucket capacity $\theta(t_k)$ is higher than $\kappa + \delta$, and hence, more packet promotion of incoming traffic could be assumed.

Zone	Condition
Z_1	$\theta(t_k) \leq \kappa - \delta$
Z_2	$\kappa - \delta \leq \theta(t_k) \leq \kappa + \delta$
Z_3	$\kappa + \delta \leq \theta(t_k)$

Table 4.2: Token bucket operating zones.

The selection of the token bucket parameters depends on the κ and δ values and the traffic that reaches the QoS system. As it is depicted in Figure 4.3, by setting $\kappa = 40$ and $\delta = 20$, the main token bucket operating zone is Z_3 for less than 6 new connections per second in the system. The more time the token bucket is in zone Z_1, the more congested the system is. That means that when more than 12 connections per second arrive to the system, it is congested most of the time.

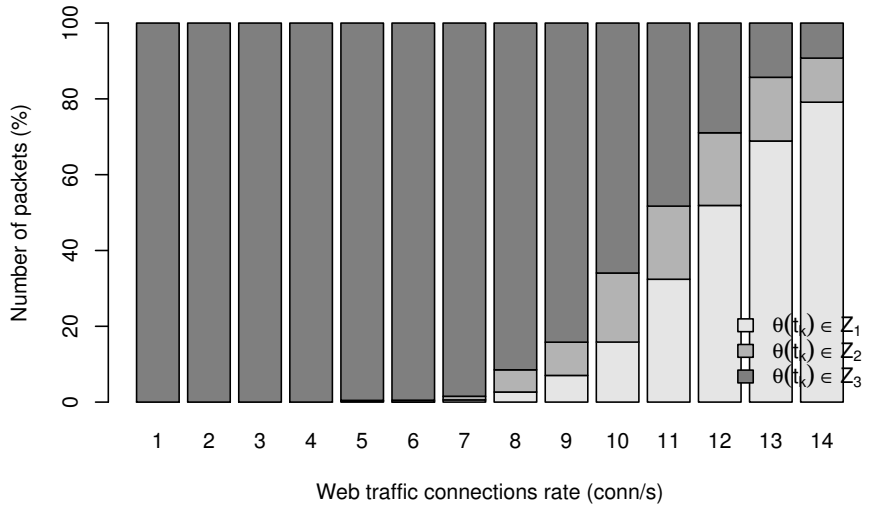


Figure 4.3: Distribution of packets in token bucket operating zones

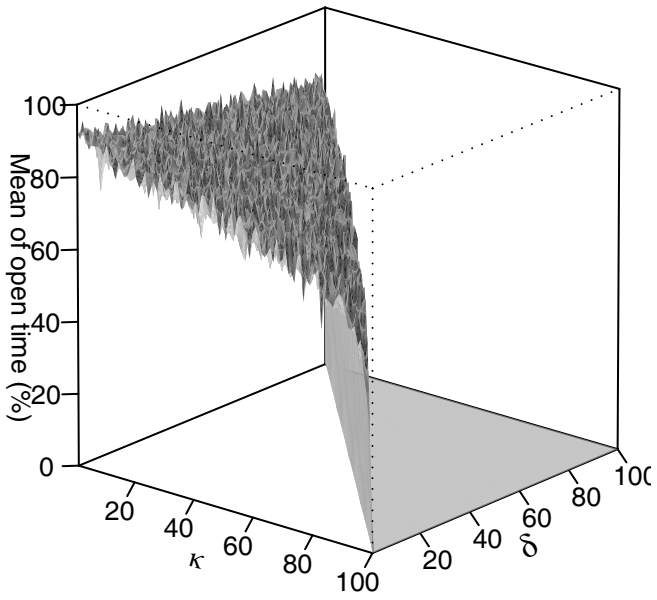


Figure 4.4: Token bucket activity for κ and δ values.

The token bucket *open/close* state depends on several factors: incoming incoming traffic, ω, κ and δ. As it has been explained above, the parameters which regulate

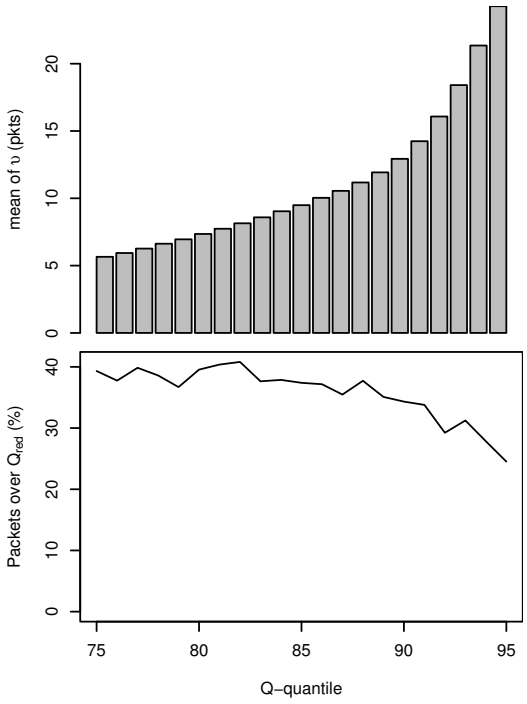


Figure 4.5: Results of the *u-quantile* over the *elephants* detection and υ value.

the filling of the token bucket are $\kappa \in [0, 100]$ and $\delta \in [0, 100]$. κ parameter is defined as the bucket level that can be filled with tokens, and δ is defined as the fluctuation bound around κ. They have to be configured as average values. The activity of token bucket for a medium congestion level is shown in Figure 4.4. From this figure, we have chosen $\kappa = 40$ and $\delta = 20$, which means that the token bucket is in *open* state when $\chi \in [20, 60]$.

The last step in the parameters tunning is the election of a suitable $u - quantile$ over $X_{t_k,H}$. The correct elephants detection in S_3 will be dependant on the election of u, that is obtained through simulation, taking into account several considerations:

- If u is too high, the threshold to differentiate between *long* and *elephant* flows will be a high value. The consequence is that only few flows will be considered as elephants. Therefore, they will be basically classified as long flows, and hence, forwarded to the medium priority queue (Q_{yellow}), instead of the lowest priority queue (Q_{red}). Consequence: elephant flows will spend bandwidth which should

be assigned to other priority flows such as long, and maybe, short flows.

- On the other hand, if u is too low, the threshold to differentiate long and elephant flows is a low value and many flows are considered as *elephant*, and therefore they are forwarded to the lowest priority queue (Q_{red}). Consequence: certain flows which should be treated as long, and promoted to the highest priority queue, might be relegated to the lowest priority queue, and in consequence, penalised.

We have obtained the amount of packets that are classified as $P_{\downarrow}$ and, hence, the amount of flows which are penalised in the medium congestion level, with the same network parameters, varying u. Results are depicted in Figure 4.5. We have obtained the results setting $u_{quantile}$ in the $[75, 99]$ range. Once the simulation has concluded, and all the flows have been classified according the $u-quantile$, the value υ indicates the mean of flow sizes in the computed quantil. For example:

- For $u = 90$, we get a value of $\upsilon = 17\, packets$, and the 35% of long flows are marked as *elephants*.
- For $u = 95$, we get a value of $\upsilon = 30\, packets$ and only the 30% of long flows are marked as *elephants*.
- For $u = 99$, we get a value of $\upsilon = 123\, packets$ and only the 20% of long flows are marked as *elephants*.

We have decided to adopt $u = 95$ as the threshold to differentiate between long and elephant flows.

4.3 Results for S_1, $S_1 \oplus S_2$ and $S_1 \oplus S_2 \oplus S_3$

The first step we have considered is to obtain the performance results for each stage of LFP, in a progressive way. In this section, we present the results of simulating S_1, then, results for $S_1 \oplus S_2$ and finally, results of $S_1 \oplus S_2 \oplus S_3$ simulation.

As it was specified in the previous chapter, the packets classification is based on marking and forwarding over different queues processes. Each stage introduces different labels which are used to mark incoming packets. For that reason, first of all, we need to asses the effect of the S_1, S_2 and S_3 of LFP algorithm. The packets classification for each stage is depicted in Figure 4.6.

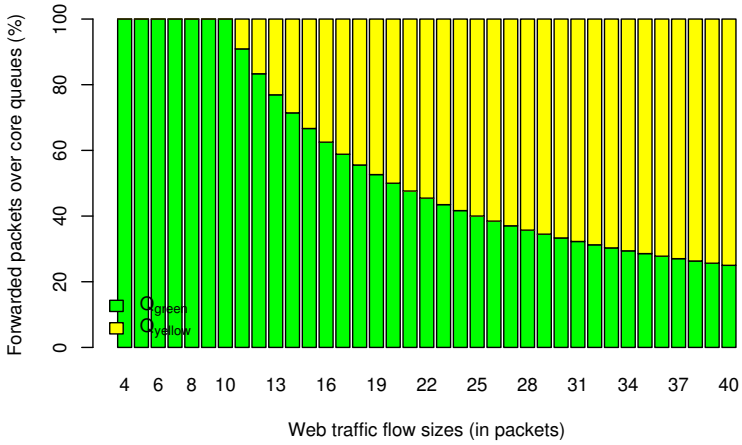
As it is illustrated in Figure 4.6(a), the S_1 stage uses the label set defined by $L_1 = \{P_S, P_L\}$. This stages also establishes a hard limit between short and long flows. This limit is defined by $\tau = 13\,packets$, therefore, every packet in the range $[1, \tau]$ is classified as short, and marked as P_S. The rest of packets in the range $[\tau + 1, \chi^f(t_k)]$ are marked as P_L because they are classified as long flows.

Considering the second stage of the algorithm, the behaviour of LFP by using $S_1 \oplus S_2$ is illustrated in Figure 4.6(b). This stage uses the label set defined by $L_2 = \{P_S, P_L, P_\uparrow\}$ where the label $P_\uparrow$ has been added in order to identify the promoted packets. In this stage, some packets that will be condemned to the lowest priority queue are recovered and promoted to receive the same treatment as short flows, in order to improve the final performance of flows with sizes close to τ.

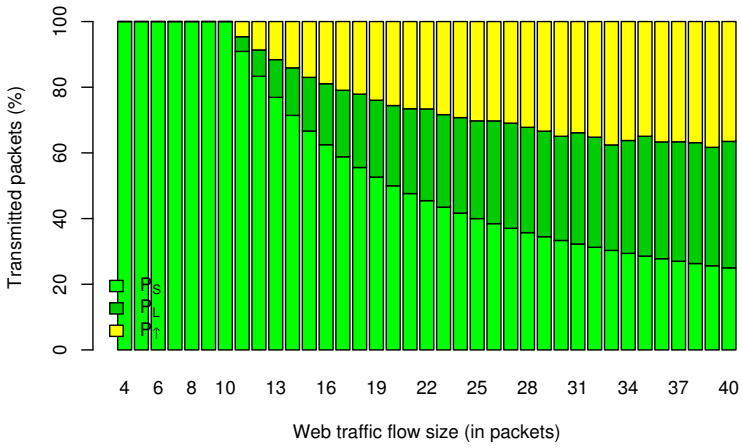
Finally, the last stage that considers $S_1 \oplus S_2 \oplus S_3$ constraints is depicted in Figure 4.6(c). This stage uses the labels set defined by $L_2 = \{P_S, P_L, P_\uparrow, P_\downarrow\}$ where the label $P_\downarrow$ is used to mark those packets that have been treated as elephants, and hence, forwarded to the lowest priority queue. An immediate consequence of this fact is the releasing bandwidth in Q_{yellow} and therefore, there is more bandwidth to promote long flows to the Q_{green} queue.

We have measured the performance parameters for S_1, S_2 and S_3 stages by simulation. We show the results in Table 4.3 for the different scenarios already defined in Table 4.1. The metrics shown are the link utilization, throughput and goodput, and also other custom parameters that we have defined as: P_{drop} and F_{drop}. The drop packet probability P_{drop} provides the dropped packet rate detected in each situation and F_{drop} provides the probability of flows having some dropped packet. Obviously, we are looking for strategies that maintain both probability measures as the lowest.

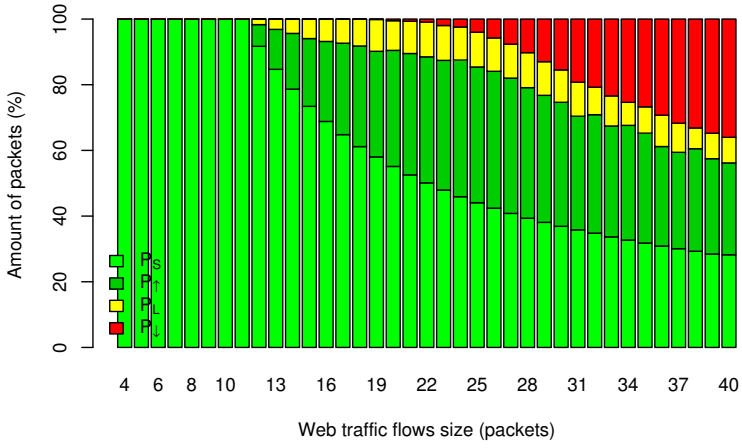
We have to analyse the performance parameters together. From this point of view, the utilization and throughput values are similar in the three configurations



(a) LFP S_1.



(b) LFP $S_1 \oplus S_2$.



(c) LFP $S_1 \oplus S_2 \oplus S_3$.

Figure 4.6: Packets classification for the S_1, S_2 and S_3 stages in LFP.

Level	*Alg*	*U*	*Throughput*	*Goodput*	*Overhead*	P^{drop}	F^{drop}
Low	S_1	9.88	0.208	0.198	4.96	3.99e-04	1.18e-03
	$S_1 \oplus S_2$	10.6	0.223	0.212	4.99	5.03e-04	1.38e-03
	$S_1 \oplus S_2 \oplus S_3$	10.7	0.226	0.215	4.96	2.16e-04	6.44e-04
Medium	S_1	16.0	0.338	0.321	5.08	1.26e-03	3.33e-03
	$S_1 \oplus S_2$	15.0	0.317	0.301	5.05	1.15e-03	3e-03
	$S_1 \oplus S_2 \oplus S_3$	16.1	0.339	0.322	4.99	5.88e-04	1.76e-03
Heavy	S_1	21.5	0.454	0.431	5.23	3.32e-03	8.2e-03
	$S_1 \oplus S_2$	21.3	0.45	0.427	5.16	2.38e-03	5.92e-03
	$S_1 \oplus S_2 \oplus S_3$	21.4	0.451	0.428	5.1	1.81e-03	5.17e-03

Table 4.3: Summarised values for the different LFP stages.

(S_1, S_2 and S_3), but we can emphasise the best result in goodput, overhead, P_{drop} and F_{drop} is achieved by the final LFP version (composed by S_1, S_2 and S_3 stages) . As we have mentioned above, it is necessary to analyse these measures in a joint manner, therefore, we highlight the results for medium congestion scenario, where every performance parameter is improved in the LFP final stage.

4.4 Comparative with other proposals

In this section we are going to provide the simulation results for our algorithm LFP in the final version with the three stages (S_1, S_2 and S_3). We have compared LFP results with simulation results obtained by Droptail, RED, and SFD. The arguments for this comparative are detailed below:

1. We want to compare LFP with a non-active queue management algorithm, just a queue management algorithm such as Droptail.

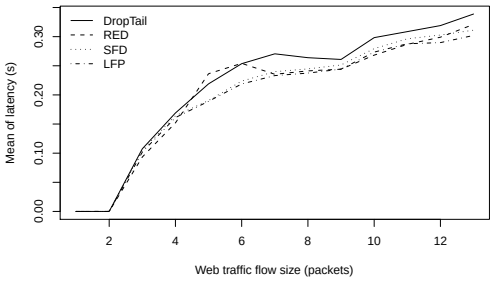
2. We have compared it with RED because this is the most implemented AQM strategy in switched packet networks.

3. And finally, we have compared it with SFD because it is the starting point of our proposal.

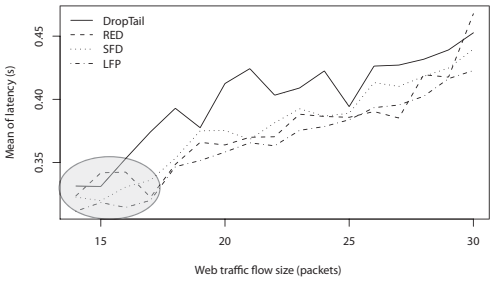
We have simulated the four algorithms (DropTail, RED, SFD and LFP) under the same topology and network parameters. We have also simulated every algorithm in the congestion scenarios defined in Table 4.1 and, finally, we have separated the obtained measures in two sets: for short and for long flows. We propose this classification in order to provide more detailed values for the performance parameters that we have analysed.

The first comparative is regarding the end-to-end latency. In this case, we have classified the total of flows in two sets: short and long flows. We have measured the latency in the three congestion scenarios previously described that we have proposed: low, medium and heavy congestion. Results are shown in Figure 4.7.

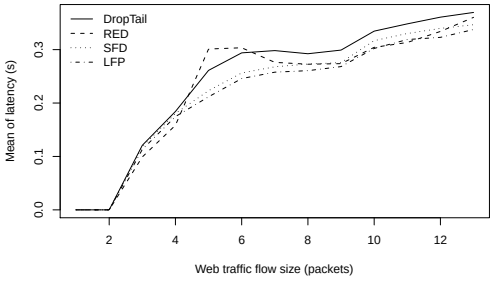
Considering short flows, we observe similar results in low and medium congestion scenarios. However, the behaviour of Droptail gets worse as the congestion increases and its end-to-end latency increases till values close to half a second. Considering now long flows, we still observe a significant increase in the latency perceived in DropTail. Moreover, we can extract from the figure the improvement of latency values in LFP compared to RED and SFD for every congestion levels when the flow size is close to the threshold (marked with a circle in Figure 4.7) where LFP outperforms in terms of latency the rest of algorithms.



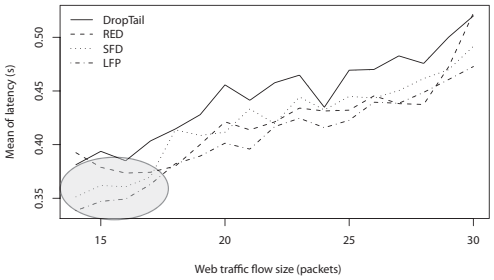
(a) Low congestion: latency



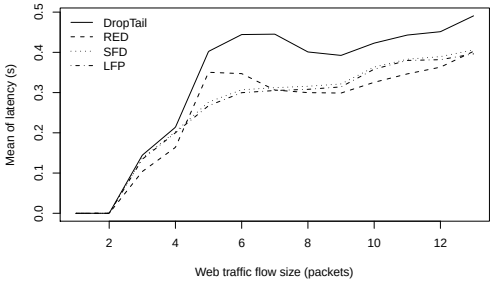
(b) Low congestion: latency



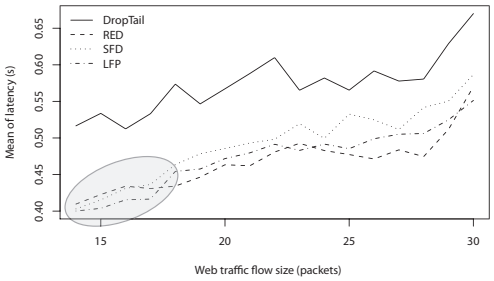
(c) Medium congestion: latency



(d) Medium congestion: latency



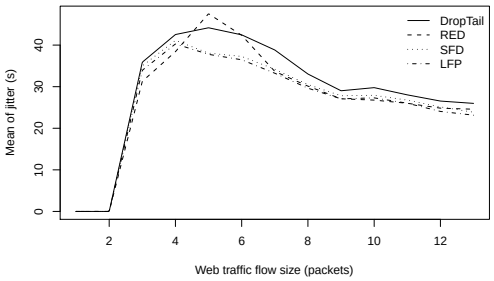
(e) Heavy congestion: latency



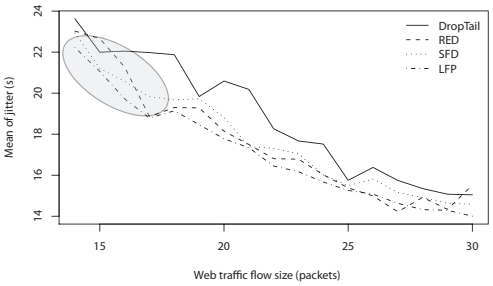
(f) Heavy congestion: latency

Figure 4.7: Latency for short and long flows for different congestion scenarios.

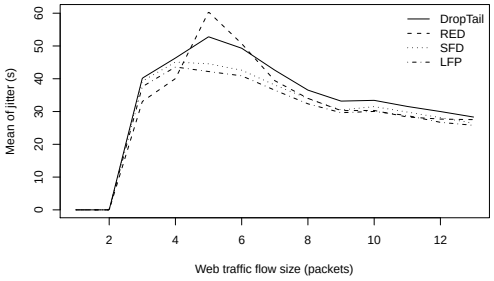
Let us analyse jitter results that shows a behaviour similar to the latency for each simulated algorithm. As we can observe in Figure 4.8, DropTail gets again the worst results for every congestion level and for short and long flows. LFP obtains similar measured values in short flows for every congestion level but it gest less jitter than the other simulated options when considering flow sizes close to the threshold that differentiates long flows. We have marked this effect with a circle in Figure 4.8.



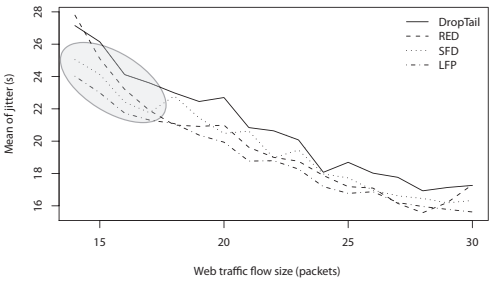
(a) Low congestion: jitter



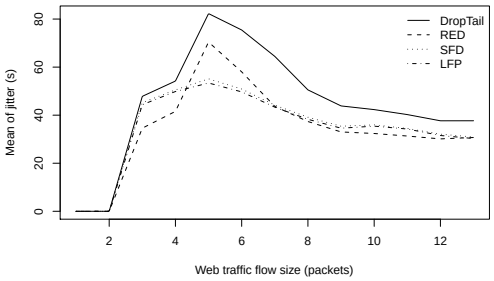
(b) Low congestion: jitter



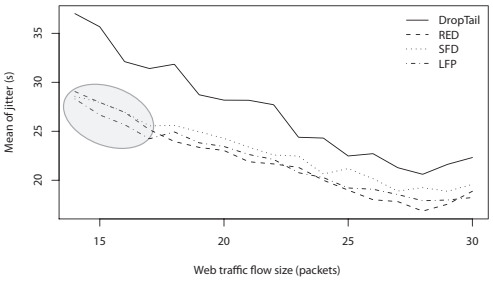
(c) Medium congestion: jitter



(d) Medium congestion: jitter



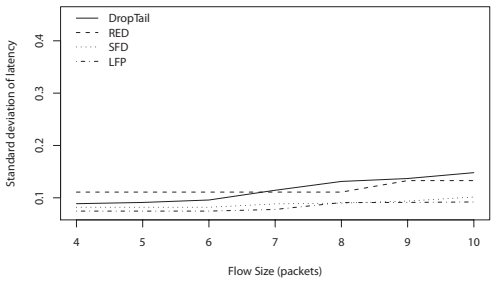
(e) Heavy congestion: jitter



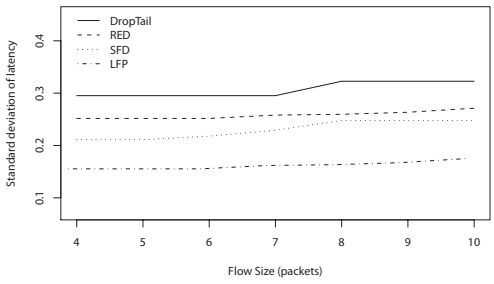
(f) Heavy congestion: mean of jitter

Figure 4.8: Jitter for short and long flows and different congestion scenarios.

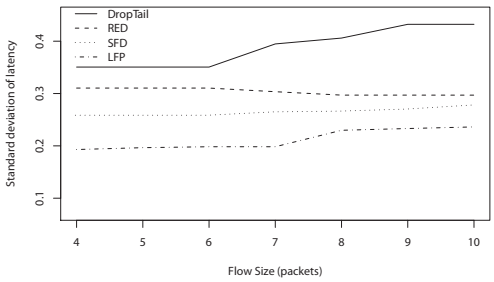
We have also computed the standard deviation for latency results. This value provides the dispersion level of the measured values, in this case, the latency. We consider that it is another factor that has an influence in the final QoS of the system. A high value for the standard deviation indicates a high level of dispersion, that means that measures are quite different. In contrast, a low value for standard deviation indicates a low level of dispersion in the measures for latency. In our case, Figure 4.9 shows the standard deviation only for short flows, and as we can observe in the figure, LFP gets the best behaviour for the three congestion levels.



(a) Low congestion: latency



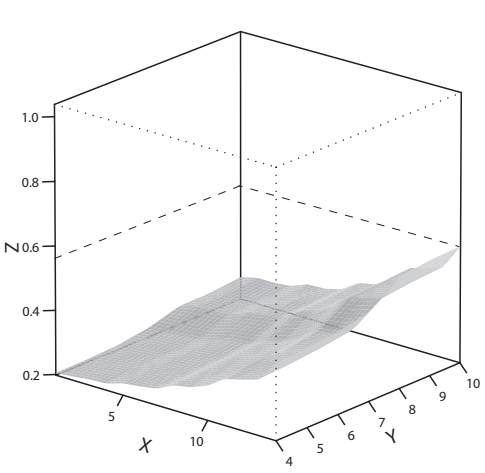
(b) Medium congestion: latency



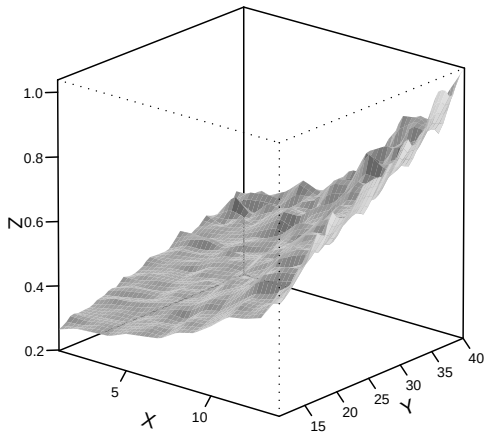
(c) Heavy congestion: latency

Figure 4.9: Standard deviation only for short flows and different congestion scenarios.

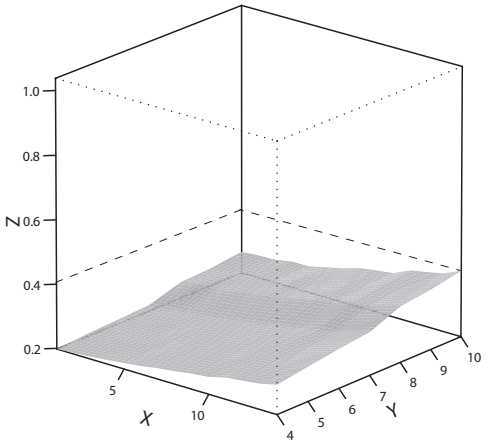
We consider that a low value of latency's standard deviation is a desired goal for every algorithm that pretends to improve the QoS of a certain Internet service, as it permits to easily guarantee a value of a SLA in terms of latency.



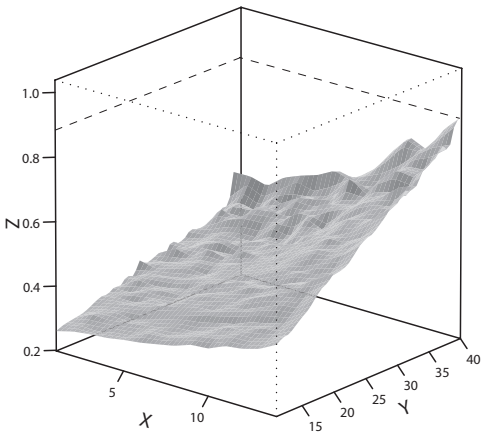
(a) DropTail: short ows



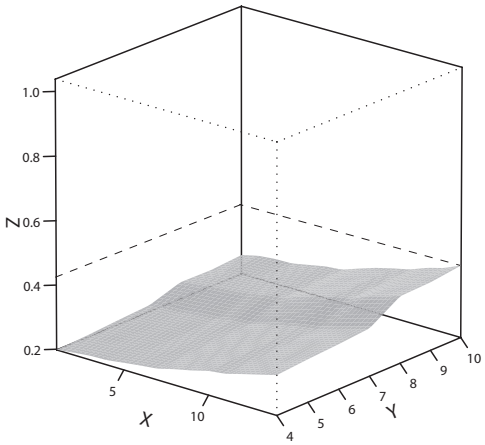
(b) DropTail: long ows



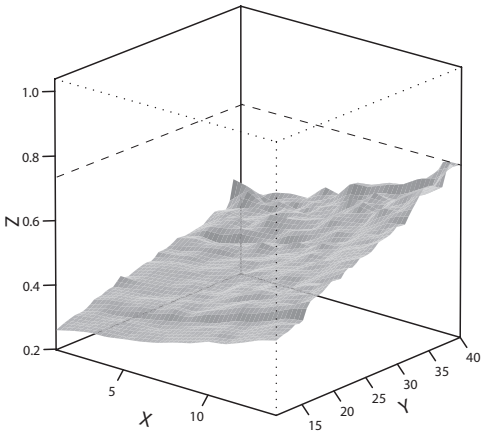
(c) RED: short ows



(d) RED: long ows



(e) LFP: short ows



(f) LFP: long ows

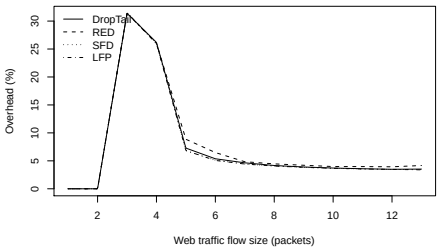
Figure 4.10: 3D graphics for short and long flows, where X-axis is the web traffic load (new conn/s), Y-axis is the flow size (packets) and Z-axis (seconds) is the smoothed mean of latency.

The end-to-end final latency has been plotted for each algorithm comparing the evolution of the web traffic load, web flow size and latency. For this reason, it has been used a 3D graph in Figure 4.10. The results have also been compared by differentiating between short and long flows. Obviously, a 3D graph shows a smoothed surface where the details are hidden, but it provides the trend of the general behaviour of the system. The X-axis shows the web traffic load, from 0 to 14 conn/s. The Y-axis shows the flow size in packets. In the case of short flows, Y-axis shows from 1 to τ packets, and for long flows, Y-axis goes from $\tau + 1$ to 40 packets. And finally, Z-axis shows the mean latency in seconds. Since the plotted surface shows the latency results (Z-axis) for different web traffic load (X-axis) for different web flow size (Y-axis), from the point of view of the surface appearance, DropTail shows some spikes and a steeper slope, specially at heavy congestion level and for long flow sizes. By contrast, LFP gets a more uniform surface for latency in both cases, short and long flows. These results have been summarised in numerical values by computing the mean of latency, with 0.95 confidence intervals, for short and long flows (Table 4.4).

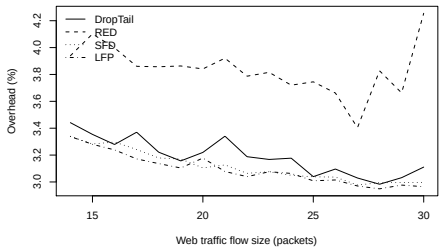
	Short flows	Long flows
DropTail	0.30 [0.28, 0.31]	0.48 [0.46, 0.49]
RED	0.26 [0.25, 0.27]	0.45 [0.43, 0.46]
LFP	0.27 [0.26, 0.28]	0.43 [0.42, 0.44]

Table 4.4: Summarised data with 0.95 confidence intervals for the overall latency.

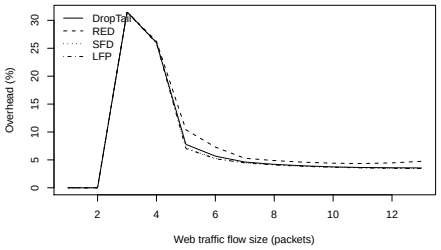
We have also computed the overhead for each scenario classified by short and long flows. The results are shown in Figure 4.11, where we can observe a very similar behaviour for every algorithm for short flows. Regarding long flows, we observe the worst overhead measure is obtained by RED algorithm, and we emphasise the best behaviour for the LFP algorithm.



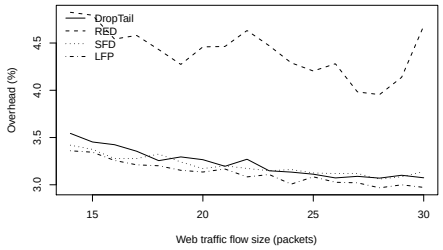
(a) Low congestion: overhead



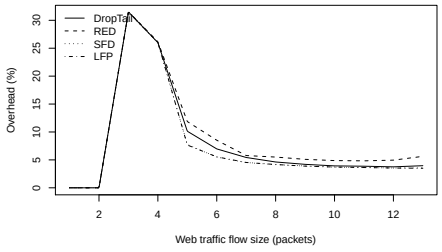
(b) Low congestion: overhead



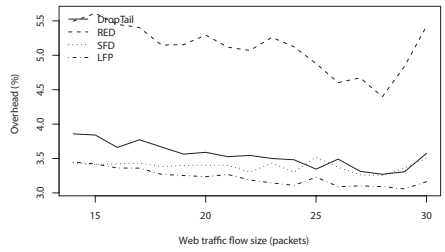
(c) Medium congestion: overhead



(d) Medium congestion: overhead



(e) Heavy congestion: overhead



(f) Heavy congestion: overhead

Figure 4.11: Overhead for short and long flows and different congestion scenarios.

We want to point out the fact that our proposal is the one that gets the lowest overhead in all the tested scenarios. This means that the system needs less effort to get similar performance as the one obtained by well known algorithms as RED or SFD.

Level	*Alg*	*U*	*Throughput*	*Goodput*	*Overhead*	P^{drop}	F^{drop}
Low	DropTail	10.1	0.213	0.202	5.06	1.44e-03	3.49e-03
	RED	10.5	0.222	0.210	5.66	7.63e-03	1.99e-02
	SFD	9.88	0.208	0.198	4.96	3.99e-04	1.18e-03
	LFP	10.7	0.226	0.215	4.96	2.16e-04	6.44e-04
Medium	DropTail	16.2	0.341	0.323	5.11	2.25e-03	6.12e-03
	RED	15.7	0.335	0.315	6.2	1.34e-02	3.5e-02
	SFD	16.0	0.338	0.321	5.08	1.26e-03	3.33e-03
	LFP	16.1	0.339	0.322	4.99	5.88e-04	1.76e-03
Heavy	DropTail	21.2	0.449	0.425	5.43	7.43e-03	1.95e-02
	RED	20.9	0.449	0.418	6.81	2.00e-02	5.13e-02
	SFD	21.5	0.454	0.431	5.23	3.32e-03	8.2e-03
	LFP	21.4	0.451	0.428	5.1	1.81e-03	5.17e-03

Table 4.5: Summarised values for DropTail, RED, SFD and LFP.

We have also summarised every measure in Table 4.5, where we can globally compare the behaviour of DropTail, RED, SFD and LFP.

The performance results for the parameters that we have measured must be analysed together. As we can observe, LFP gets the lowest values of the P_{drop} and F_{drop} parameters. Therefore, the number of dropped packets is improved with our proposal. Comparing it with other algorithms, and under the same scenario, we can emphasise that it also improves other performance parameters, such as overhead and throughput. Regarding the latency, we can observe that LFP get a comparable result with RED and SFD for short flows, and it also improves the results for long flows. Considering the latency, LFP always improves the standard deviation and jitter for short and long flows.

Chapter 5

Conclusions and future work

5.1 Conclusions

We have proposed a DiffServ-based algorithm, named Long Flow Promotion (LFP), to improve the overall performance of web traffic. The development of the algorithm has been conducted by several premises: reduce the latency, provide a preferential treatment for short flows, permit the coexistence of short and long flows without excessive penalisation for any of them, and finally, isolate extremely long flows. The algorithm uses traditional elements of the DiffServ architecture but introduces some new characteristics as three colours processing, the Token Bucket Model, the packet promotion term, and finally, the packet penalisation.

In order to get the goodness of LFP, it has been compared with other well-known proposals, that we justify below:

1. DropTail, because we want to compare LFP with a simple queue mechanism, without nor active queue management neither QoS feature.

2. RED, as a widespread used algorithm that also implements QoS in the DiffServ environment.

3. SFD, because we have used this algorithm as our working start point.

First of all, we have justified our step-by-step final version through three stages

(S_1, S_2 and S_3). The target of S_1 is the flow differentiation. We have already layed out the main drawback of this stage, which is the penalisation of certain not so long flows. This penalisation justifies the second stage (S_2), which main target is the promotion of some flows, from low to high preferential treatment. In this stage, we have used the token bucket model as a packet promotion mechanism. In this stage, we have also mentioned another drawback, which is the existence of extremely long flows, and for that reason, we consider the third stage (S_3), where we establish an isolating and detecting mechanism for those extremely long flows. Once they have been detected, they are marked with the appropriate label and forwarded over the lowest priority queue.

The results have been obtained through simulation in ns-2, and the incoming synthetic web traffic has been generated by PackMime-HTTP. After the simulation some performance parameters related to the QoS have been analysed: latency, dropped packets, throughput and overhead.

We can conclude with some statements:

- DropTail gets the worst results becauses it is the simplest solution compared with the other algorithms. For that reason, it is not appropriate as a solution for developing QoS into the web service providers.

- RED is the most implemented algorithm in packet switching devices and it represents the main support for the QoS in service provider systems. However, our results show that LFP achieves comparable values for latency than RED, while producing smaller values of overhead and better values for throughput and less dropped packets.

- When comparing LFP with SFD, for short flows, SFD normally gets the best latency, but LFP improves the rest of performance parameters: that means better latency for long flows, throughput and link utilisation.

Considering the questions that were contemplated in the introduction of this thesis, we can conclude as follows:

How could we avoid a penalisation of flows whose size is close to the threshold?

With the token bucket features we have avoided the penalisation of those flows that are close to the size threshold by marking packets belonging to them with green colour and promoting them in the edge device and therefore, achieving a better treatment at the core device. As we have detailed previously, the performance results for short flows using our proposal (LFP) is comparable to SFD and RED, and for long flows, LFP does not suffer any significant decrease in the performance results. These better results for delay are achieved while keeping the overhead level with LFP lower than the overhead of the rest of algorithms.

How many flows can we promote while keeping a suitable system performance?

The positive results for LFP have been obtained using the promotion mechanism detailed in stage S_2. Moreover, the amount of promoted packets sent over each queue has been justified in Figure 4.6(c).

What happens if we are promoting some flows which do not need to be promoted? To avoid this fact, we have used another threshold, defined as $u_{quantile}$. The target of this threshold is to delimit long and extremely long flows. As we have demonstrated through simulation, when forwarding these flows over the lowest priority queue, LFP achieves the best results in terms of performance than the other algorithms.

How can we make the extremely long flows threshold adaptive? We have used the last history of incoming traffic to the QoS system, to adaptively set the threshold depending on the characteristics of the last flows in the system.

Finally, we can conclude that LFP can be taken into consideration as a good choice for flow differentiation in a QoS system architecture.

5.2 Future work

Regarding the future work and open research issues, we propose to develope the following ideas in a close future:

- To update the flow size thresholds tunning of the LFP algorithm with recent traffic characteristics that include the Ajax protocol or IPTV traffic.
- To redefine the token bucket model in order to deal with heavy bursty workload. It would be important to analyse in depth the repercusion of including high levels of burstiness in the workload and analyse variations of the proposed algorithm that permit the system to maintain the desired QoS.
- Before the final implementation, it would be interesting to validate the model using modern model checking technologies, such as Specification and Description Language (SDL) or Promela and Spin verification

Chapter 6

Contributions

The list of contributions and paper generated by this thesis, below:

- Alcaraz, S., Juiz, C., Gilly K. and Puigjaner, R., A New Token Bucket DiffServ Policy for Web Traffic, Proc. of the International Conference on Telecommunications & Multimedia (TEMU 2006), ISBN: 960-88785-2-7, Vol.: S2, pp:1-8, July, 2006, Crete, Greece.

- Alcaraz, S., Juiz, C., Gilly K. and Puigjaner, R., Una Estrategia de QoS para tráfico HTTP Basada en Políticas Token Bucket, Actas de la XXXII Congreso Latinoamericano de Informática, ISBN: 956-303-028-1 , pp: 1-12, August, 2006, Universidad de Santiago, Santiago, Chile.

- Alcaraz, S., Gilly, K., Juiz, C. and Puigjaner, R., A Token Bucket Model with Assured Forwarding for Web Traffic, Proc. of the NBis'07, First International Conference on Network-Based Information Systems, ISBN: 978-3-540-74572-3, ISSN: 0302-9743, Vol: 4658, pp: 298-307, September, 2007, Springer-Verlag, Heidelberg, Germany.

- Alcaraz, S., Gilly, K., Juiz, C. and Puigjaner, R., Handling HTTP Flows Over a DiffServ Framework, Proc. of the 4th international IFIP/ACM Latin American conference on Networking, ISBN 978-1-59593-907-5, Vol: 1, pp: 95-101, October, 2007, ACM, San José, Costa Rica.

- Alcaraz, S., Gilly, K., Juiz, C. and Puigjaner, R., Promoting Web Traffic Over a DiffServ Architecture, Proc. of 22nd International Symposium on Computer and Information Sciences, ISCIS 2007, ISBN: 1-4244-1364-8, Vol: 1, pp: 8-13, November 7-9, 2007, Ankara, Turkey.

- Alcaraz, S., Gilly, K., Juiz, C. and Puigjaner, R., Accommodating short and long web traffic flows over a DiffServ architecture, LNCS, ISBN: 978-3-642-24748-4, ISSN: 0302-9743, Vol: 6977, pp: 14-28, 2011, Elsevier, Heidelberg, Berlin.

Chapter 7

Appendix

7.1 NS2 model

The implementation of NS2 models has two parts of source code:

- Scripts based on Tcl.
- Source code based on C++

As you can see below, we have developed a configution file on tcl programming language. With this model, we can execute the simulation for each algorithm we have analysed in the thesis: DropTail, RED, SFD and LFP, by using the appropriate parameters in an external data file named `config.dat`.

```
#::::::::::::::::::::::::::::::::::::::::::
#::: web.tcl v1.0 (web traffic)
#::::::::::::::::::::::::::::::::::::::::::
#::::::::::::::::::::::::::::::::::::::::::
#::: reading 'config.dat' file
#::::::::::::::::::::::::::::::::::::::::::
set infile [open "config.dat" r]
while { [gets $infile read_line] >= 0 } {
    regsub -all {\'} $read_line "\"" line
    set parameter [split $line =]
    set par_name [lindex $parameter 0]
    set par_value [lindex $parameter 1]
    set cmd "set $par_name $par_value"
    eval $cmd
```

```
    puts $cmd
}
close $infile

#::::::::::::::::::::::::::::::::::::::::::
#::: reading command-line parameters
#::::::::::::::::::::::::::::::::::::::::::
set vars {FileName Algorithm Scenario CIR CBS TraceMIME
          TraceNS TracePolicy Qn Version CBRrate Kappa Delta}

set n_vars [llength $vars]

for {set i 0} {$i < $n_vars} {incr i} {
    set var_name [lindex $vars $i]
    set var_value [lindex $argv $i]
    set cmd {set $var_name $var_value}
    eval $cmd
}

#::::::::::::::::::::::::::::::::::::::::::
# useful variables
set duration [expr $Warmup + $Length]; # total simulation time (s)
set CLIENT 0
set SERVER 1

#::::::::::::::::::::::::::::::::::::::::::
#: Setup Simulator
#::::::::::::::::::::::::::::::::::::::::::
remove-all-packet-headers
add-packet-header IP TCP
set ns [new Simulator]
$ns use-scheduler Heap

#::::::::::::::::::::::::::::::::::::::::::
#:Topology
#::::::::::::::::::::::::::::::::::::::::::
#                            Edge     Core
#                             |       |
#                          B  |   A   |
#  ServersCloud <---> R3 ---> R2 ---> R1 <---> ClientsCloud
#                      ^               |
#                      |      C        |
#                      +---------------+
#::::::::::::::::::::::::::::::::::::::::::

#::::::::::::::::::::::::::::::::::::::::::
```

```
61 # link parameters
62 #:::::::::::::::::::::::::::::::::::::::::::
63 set ABandwidth "2Mb"
64 set ADelay "5ms"
65 set BLength "50"
66 set BBandwidth "10Mb"
67 set BDelay "5ms"
68 set CLength "50"
69 set CDelay "5ms"
70 set CBandwidth "10Mb"
71 set REDMinTh "10"
72 set REDMaxTh "40"
73 set REDWeight "0.002"
74 set ThresholdQCoreA "10"
75 set MaxThresholdQCoreA "30"
76 set WeightQCoreA "0.02"
77 set ThresholdQCoreB "8"
78 set MaxThresholdQCoreB "24"
79 set WeightQCoreB "0.02"
80 set ThresholdQCoreC "8"
81 set MaxThresholdQCoreC "24"
82 set WeightQCoreC "0.02"
83
84 #:::::::::::::::::::::::::::::::::::::::::::
85 # create nodes
86 set ClientCloud [$ns node]
87 set ServerCloud [$ns node]
88 set R1          [$ns node]
89 set R2          [$ns node]
90 set R3          [$ns node]
91
92 #:::::::::::::::::::::::::::::::::::::::::::
93 # create links
94 $ns duplex-link $ClientCloud $R1 10Mb 5ms DropTail
95 $ns simplex-link $R1 $R3 $CBandwidth $CDelay DropTail
96 $ns queue-limit $R1 $R3 $CLength
97 $ns duplex-link $R3 $ServerCloud 10Mb 5ms DropTail
98
99 #:::::::::::::::::::::::::::::::::::::::::::
100 # Setup a UDP connection
101 set udp [new Agent/UDP]
102 $ns attach-agent $ServerCloud $udp
103 set null [new Agent/Null]
104 $ns attach-agent $ClientCloud $null
105 $ns connect $udp $null
106
```

```
#:::::::::::::::::::::::::::::::::::::::::::
# Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set rate_ "[expr $CBRrate + 0]Mb"
$cbr set random_ false

#:::::::::::::::::::::::::::::::::::::::::::
# SEGMENTO DE CÓDIDO ESPECÍFICO PARA CADA MODELO
if { $Algorithm == "DropTail" } {# Definición de política DropTail
    $ns simplex-link $R3 $R2 $BBandwidth $BDelay DropTail  ;
    $ns queue-limit $R3 $R2 $BLength

    $ns simplex-link $R2 $R1 $ABandwidth $ADelay DropTail  ;
    $ns queue-limit $R2 $R1 $QLength
} elseif { $Algorithm == "RED" } { # Definición de política RED
    $ns simplex-link $R3 $R2 $BBandwidth $BDelay RED  ;
    $ns queue-limit $R3 $R2 $BLength

    $ns simplex-link $R2 $R1 $ABandwidth $ADelay RED  ;
    $ns queue-limit $R2 $R1 $QLength

}  elseif { $Algorithm == "SFD" || $Algorithm == "LFP"}  { # SFD/PLF
    $ns simplex-link $R3 $R2 $BBandwidth $BDelay dsRED/edge  ;
    $ns queue-limit $R3 $R2 $BLength

    $ns simplex-link $R2 $R1 $ABandwidth $ADelay dsRED/core  ;
    $ns queue-limit $R2 $R1 $QLength

    #:::::::::::::::::::::::::::::::::::::::::::
    #: Setup Diffserv parameters
    #:::::::::::::::::::::::::::::::::::::::::::
    set qEdge [[$ns link $R3 $R2] queue]
    set qCore [[$ns link $R2 $R1] queue]

    #:::::::::::::::::::::::::::::::::::::::::::
    # setup EDGE node parameters
    $qEdge meanPktSize $MSS
    $qEdge set numQueues_ 1
    $qEdge setNumPrec 3

    if {$Algorithm == "SFD"} {
        set Version 1 }

    $qEdge addPolicyEntry \
        -1 -1 LFP 10 $SizeThreshold $CIR $CBS $Version \
```

```
153         $TracePolicy $ACK $RTX $HistoryQn $Qn $SimultFlowsW $Kappa $Delta
154
155     $qEdge addPolicerEntry LFP 10 11 12
156     $qEdge addPHBEntry 10 0 0
157     $qEdge addPHBEntry 11 0 1
158     $qEdge addPHBEntry 12 0 2
159
160     #::::::::::::::::::::::::::::::::::::::::::
161     # setup CORE node parameters
162     $qCore setMREDMode RIO-C
163     $qCore setSchedularMode PRI
164
165     $qCore meanPktSize $MSS
166     $qCore set numQueues_ 1
167     $qCore setNumPrec 3
168
169     $qCore addPHBEntry 10 0 0
170     $qCore addPHBEntry 11 0 1
171     $qCore addPHBEntry 12 0 2
172
173     $qCore configQ 0 0 $ThresholdQCoreA $MaxThresholdQCoreA $WeightQCoreA
174     $qCore configQ 0 1 $ThresholdQCoreB $MaxThresholdQCoreB $WeightQCoreB
175     $qCore configQ 0 2 $ThresholdQCoreC $MaxThresholdQCoreC $WeightQCoreC
176 }
177
178 #::::::::::::::::::::::::::::::::::::::::::
179 # Setup queue parameters (if it's necesary)
180 #::::::::::::::::::::::::::::::::::::::::::
181 # setup TCP
182 Agent/TCP/FullTcp set segsize_ $MSS;
183 Agent/TCP set window [expr round($Window*1024.0/($MSS + 40.0))]
184
185 #::::::::::::::::::::::::::::::::::::::::::
186 #: Setup PackMime
187 #::::::::::::::::::::::::::::::::::::::::::
188 set pm [new PackMimeHTTP]
189 $pm set-client $ClientCloud;
190 $pm set-server $ServerCloud;
191 $pm set-rate $Scenario;           # new connections per second
192
193 #::::::::::::::::::::::::::::::::::::::::::
194 # Setup PackMime Random Variables
195 global defaultRNG
196
197 # create RNGs (appropriate RNG seeds are assigned automatically)
198 set semilla 0
```

```
199
200 set flowRNG [new RNG]
201 $flowRNG seed $semilla
202
203 set reqsizeRNG [new RNG]
204 $reqsizeRNG seed $semilla
205
206 set rspsizeRNG [new RNG]
207 $rspsizeRNG seed $semilla
208
209 # create RandomVariables
210 set flow_arrive [new RandomVariable/PackMimeHTTPFlowArrive $Scenario]
211
212 set req_size [new RandomVariable/Uniform]
213 $req_size set min_ 100
214 $req_size set max_ 200
215
216 set rsp_size [new RandomVariable/PackMimeHTTPFileSize $Scenario $SERVER]
217
218 # assign RNGs to RandomVariables
219 $flow_arrive use-rng $flowRNG
220 $req_size use-rng $reqsizeRNG
221 $rsp_size use-rng $rspsizeRNG
222
223 # set PackMime variables
224 $pm set-flow_arrive $flow_arrive
225 $pm set-req_size $req_size
226 $pm set-rsp_size $rsp_size
227
228 # record HTTP statistics
229 set fileMime "$FileName.mime"
230 if {$TraceMIME} { $pm set-outfile $fileMime}
231
232 #::::::::::::::::::::::::::::::::::::::::::
233 # Packet Tracing
234 #::::::::::::::::::::::::::::::::::::::::::
235 proc trace {} {
236     global ns R2 R1 FileName CBRrate
237
238     # setup packet tracing
239     Trace set show_tcphdr_ 1
240
241     set fileOut1 "$FileName.tr"
242     set qmonf1 [open "|
243                       grep ^\[d\-\] | cut -d \ \" \" -f 1,2,6,8,12 > $fileOut1" w]
244
```

```
245     $ns trace-queue $R2 $R1 $qmonf1
246 }
247
248 #::::::::::::::::::::::::::::::::::::::::::
249 # Cleanup
250 #::::::::::::::::::::::::::::::::::::::::::
251 proc finish {} {
252     global ns pm reqsizeRNG rspsizeRNG flowRNG req_size rsp_size
253                                 flow_arrive TraceNS
254     if {$TraceNS} {$ns flush-trace}
255
256     # delete all of the RNGs and RanVars we created reqsizeRNG
257     delete $rspsizeRNG
258     delete $flowRNG
259     delete $req_size
260     delete $rsp_size
261     delete $flow_arrive
262
263     # delete PackMime $pm Simulator $ns
264     exit 0
265 }
266
267 #::::::::::::::::::::::::::::::::::::::::::
268 # Simulation Schedule
269 #::::::::::::::::::::::::::::::::::::::::::
270 $ns at 0.0 "$pm start"
271
272 if {$CBRrate} { $ns at 0.0 "$cbr start"}
273 if {$TraceNS} {$ns at $Warmup "trace"}
274
275 $ns at $duration "$pm stop"
276
277 if {$CBRrate} {$ns at $duration "$cbr stop"}
278
279 $ns at [expr $duration + 1] "finish"
280
281 $ns run
```

7.2 Configuration file

Different configuration parameters have been specified in `config.dat` file. We can simulates different scenarios or model properties by changing these parameters.

```
1 WebScenarios='1 2 3 4 5 6 7 8 9 10 11 12 13 14'
```

```
Scenario='3'
CongestionLevels='6 10 14'
Length='5000'
Samples='10'
SizeThreshold='12'
Rango1='4 40'
Rango2='8 15'
Pi='6'
LFPVersions='1 2 3'
ABandwidth='2'
ACK='1'
RTX='1'
Qn='98'
QnList='90 91 92 93 94 95 96 97 98 99'
CIR='500'
CIRList='500 1000 1500'
CBS='500'
CBSList='500 1000 1500'
Kappa='40'
KappaList='1 2 3 4 5 6 7 8 9'
Delta='20'
DeltaList='1 2 3 4 5 6 7 8 9'
QLength='50'
QLengthList='10 20 30 40 50 60 70'
CBRLow='0.0001'
CBRMedium='0.001'
CBRHeavy='0.01'
Warmup='0'
MSS='1460'
PacketInterval='10'
ThWindowLeft='4'
ThWindowRight='12'
RatioA='0'
RatioB='1.4'
RatioC='10'
PrecGraph='500'
PrecHist='400'
ConfidenceIntervalA='1'
ConfidenceIntervalB='0.95'
VeryLongFlow='500'
HistoryQn='500'
SimultFlowsW='0.7'
Window='16'
Condor='1'
TailA='2'
TailB='4'
```

```
48 CondorMaxJobs='0'
```

7.3 C++ source code

Model changes need to be added to the source code of the NS2 tool. In this case, we have added the LFP Policy C++ object as you can see below. This object implements every LFP property and characteristic.

```
1  QuantileClass::QuantileClass () {
2    indexIN=0;
3
4    for(int i=0;i<MAXQUANTIL;i++) {
5      historic[i]=0;
6
7      for (int j;j<3;j++)
8        U[i][j]=0;
9    }
10 }
11
12 void QuantileClass::addValuesHistoric(int v, int h) {
13   maxHistoric = h;
14
15   if (indexIN == maxHistoric)
16     indexIN=0;
17
18   historic[indexIN]=v;
19   indexIN++;
20   //  printf("index=%d\n",indexIN);
21   return;
22 }
23
24 void QuantileClass::printHistoric(void) {
25   for (int i=0;i<maxHistoric;i++)
26     if ( historic[i] )
27       printf("\n history[%d]=%d",i,historic[i]);
28   return;
29 }
30 int QuantileClass::countHistoric(void) {
31   int c=0;
32   for (int i=0;i<maxHistoric;i++)
33     if (historic[i])
34       c++;
35   return c;
```

```
36 }
37
38 void QuantileClass::printU(void) {
39   for (int i=0;i<maxHistoric;i++)
40     if ( U[i][0] )
41       printf("\n U[%d]=\t%d\t%d\t%d",i,U[i][0],U[i][1],U[i][2]);
42   return;
43 }
44
45 int QuantileClass::computeQuantile(int q) {
46   int clase, nElems, retorno;
47   int i,j;
48
49   for(i=0;i<maxHistoric;i++)
50     for (j=0;j<3;j++)
51       U[i][j]=0;
52
53   for (i=0;i<maxHistoric;i++) {
54     if (historic[i]) {
55       clase=historic[i];
56       U[clase][0]++;
57     }
58   }
59
60   for (clase=1;clase<maxHistoric;clase++) {
61     U[clase][1]=U[clase-1][1]+U[clase][0];
62     nElems=U[clase][1];
63   }
64
65   for (clase=1;clase<maxHistoric;clase++)
66     U[clase][2]=(int) 100*U[clase][1]/nElems;
67
68   retorno=0;
69   for (clase=1;clase<maxHistoric;clase++)
70     if ( U[clase][2] <= q && U[clase][0])
71       retorno = clase;
72
73   return (retorno);
74 }
75
76 //---------------------------------------
77 // Beginning of LFP
78 LFPPolicy::LFPPolicy() : Policy() {
79   flow_table.head = flow_table.tail = NULL;
80   quantilePkts=new(QuantileClass);  //  QuantileClass *quantile;
81 }
```

```
82
83 //--------------------------------------
84 LFPPolicy::~LFPPolicy(){
85   struct flow_entry *p, *q;
86   p = q = flow_table.head;
87   while (p) {
88     printf("free flow: %d\n", p->fid);
89     q = p;
90     p = p->next;
91     free(q);
92   }
93   p = q = NULL;
94   flow_table.head = flow_table.tail = NULL;
95 }
96
97 /*--------------------------------------
98  void LFPPolicy::applyMeter(policyTableEntry *policy, Packet *pkt)
99  Flow states are kept in a linked list.
100 Record how many bytes has been sent per flow and check if there is any flow
101 timeout.
102  --------------------------------------*/
103 void LFPPolicy::applyMeter(policyTableEntry *policy, Packet *pkt) {
104   int fid, srcId, dstId, founded, expired, rtx, ack;
105   struct flow_entry *flow, *new_entry, *q;
106   double ganancia,now;
107   hdr_cmn * hdr;
108   hdr_ip  * iph;
109   hdr_tcp * tcph;
110
111   now = Scheduler::instance().clock();
112
113   ganancia = (double) policy->cir * (now - policy->arrivalTime);
114   if ( ( policy->cBucket += ganancia ) > policy->cbs)
115     policy->cBucket = policy->cbs;
116
117   policy->arrivalTime = now;
118
119   hdr   = hdr_cmn::access(pkt);
120   tcph  = hdr_tcp::access(pkt);
121   iph   = hdr_ip::access(pkt);
122
123   fid   = iph->flowid();
124   dstId = iph->daddr();
125   srcId = iph->saddr();
126
127   rtx   = tcph->reason();
```

```
128  ack   = (hdr->size() == 40 ? 1 : 0);
129
130  // ----------------------------------------
131  // Revisamos la cabeza de la lista, para detectar y marcar expired flows
132  // ----------------------------------------
133  flow              = flow_table.head;
134  policy->flowsInst = 0;
135  founded           = 0;
136  expired           = 0;
137
138  while ( flow ) {
139    switch ( ( flow->last_update + FLOW_TIME_OUT < now) ? 1 : 0 ) {
140    case 0:
141      policy->flowsInst++;
142      if ( flow->fid == fid ) {
143        founded = 1;
144        flow->last_update = now;
145        flow->pkts++;
146
147        if ( !rtx && !ack )
148          flow->bytes_sent += hdr->size();
149      }
150      flow=flow->next;
151      break;
152    case 1:
153      expired=1;
154      q=flow;
155      if ( flow == flow_table.head && flow == flow_table.tail ) {
156        flow = flow_table.head = flow_table.tail = NULL;
157      } else if ( flow == flow_table.head ) {
158        flow=flow->next;
159        flow_table.head = flow;
160        flow_table.head->prev=NULL;
161      } else  if ( flow == flow_table.tail ) {
162        flow=flow->prev;
163        flow_table.tail = flow;
164        flow_table.tail->next=NULL;
165      } else {
166        flow=flow->next;
167        flow->prev=q->prev;
168        q->prev->next=flow;
169      };
170
171      quantilePkts->addValuesHistoric(q->pkts, policy->QnHistory);
172      free(q);
173      break;
```

```
174     }
175   }
176
177   // ----------------------------------------
178   // Flujo nuevo, por lo tanto, aqadir al final
179   // ----------------------------------------
180   if ( ! founded ) {
181     policy->flowsInst++;
182
183     new_entry               = new flow_entry;
184
185     new_entry->fid          = fid;
186     new_entry->src_id       = srcId;
187     new_entry->dst_id       = dstId;
188     new_entry->last_update  = now;
189     new_entry->bytes_sent   = hdr->size();
190     new_entry->count        = 0;
191     new_entry->type         = F_SHORT;
192     new_entry->pkts         = 1;
193     new_entry->next         = NULL;
194     new_entry->prev         = NULL;
195
196     if ( flow_table.tail ) { // always insert the new entry to the tail.
197       flow_table.tail->next = new_entry;
198       new_entry->prev       = flow_table.tail;
199     } else flow_table.head       = new_entry;
200
201     flow_table.tail         = new_entry;
202   }
203
204   if ( expired )
205     policy->Qn = quantilePkts->computeQuantile(policy->QnValue);
206
207   policy->flowsAvg = (int) round((1-policy->simultFlowsW) * policy->flowsAvg\
208                                  + policy->simultFlowsW * policy->flowsInst);
209   return;
210 }
211
212 /*----------------------------------------
213   void LFPPolicy::applyPolicer(policyTableEntry *policy, int initialCodePt,
214         Packet *pkt)
215 ----------------------------------------*/
216 int LFPPolicy::applyPolicer(policyTableEntry *policy,
217 policerTableEntry *policer, Packet *pkt) {
218   int fid, size, pktId, newCodePt, newPktType, ack, rtx;
219   struct flow_entry *flow;
```

```
220  hdr_ip* iph;
221  hdr_cmn* hdr;
222  hdr_tcp* tcph;
223  double now,lapsus,bucketTotalTime;
224  int BucketLevel, delta, FlowType;
225
226  now    = Scheduler::instance().clock();
227
228  // Accediendo a cabeceras y datos de paquetes  NS
229  iph    = hdr_ip::access(pkt);
230  fid    = iph->flowid();
231
232  hdr    = hdr_cmn::access(pkt);
233  size   = hdr->size();
234  pktId  = hdr->uid();
235
236  tcph   = hdr_tcp::access(pkt);
237  rtx    = (tcph->reason() ? 1 : 0);
238
239  ack    = (hdr->size() == 40 ? 1 : 0);
240
241  flow    = flow_table.head;
242  while ( flow && (flow->fid != iph->flowid()) )
243    flow = flow->next;
244
245  FlowType= (flow->pkts <= policy->th ? F_SHORT : F_LONG);
246  delta = (flow->pkts > policy->Qn) ? 1 : 0;
247  BucketLevel = (int)(100*policy->cBucket/policy->cbs);
248
249  int BucketMinLevel,BucketMaxLevel;
250
251  if ((BucketMinLevel=policy->BucketKappa - policy->BucketDelta) < 0)
252    BucketMinLevel=0;
253
254  if ((BucketMaxLevel=policy->BucketKappa + policy->BucketDelta) > 100)
255    BucketMinLevel=100;
256
257  switch ( policy->BucketOpen ) {
258  case 0:
259    if ( BucketLevel >=  BucketMaxLevel ) {
260      policy->BucketOpen = 1;
261      policy->BucketOpenTimeInit = now;
262    }
263    break;
264  case 1:
265    lapsus=now-policy->BucketOpenTimeInit;
```

```
266     if ( BucketLevel <= BucketMinLevel ) {
267       policy->BucketOpenTime += lapsus;
268       policy->BucketOpenTimeInit = 0;
269       lapsus = 0;
270       policy->BucketOpen = 0;
271     }
272     break;
273   }
274   bucketTotalTime=policy->BucketOpenTime + lapsus;
275
276   switch ( policy->version ) {
277   case 1:
278     switch (FlowType) {
279     case F_SHORT:
280       flow->type       = F_SHORT;
281       policy->cBucket = Ajusta(policy->cBucket,size);
282       newCodePt        = policer->initialCodePt;
283       newPktType       = P_Type0;
284       break;
285     case F_LONG:
286       flow->type = F_LONG;
287       newCodePt  = policer->downgrade1;
288       newPktType = P_Type2;
289       break;
290     }
291     break;
292
293   case 2:
294     switch (FlowType) {
295     case F_SHORT:
296       flow->type       = F_SHORT;
297       policy->cBucket = Ajusta(policy->cBucket,size);
298       newCodePt        = policer->initialCodePt;
299       newPktType       = P_Type0;
300       break;
301     case F_LONG:
302       flow->type=F_LONG;
303
304       switch (policy->BucketOpen) {
305       case 0:
306         newCodePt  = policer->downgrade1;
307         newPktType = P_Type2;
308         break;
309       case 1:
310         policy->cBucket = Ajusta(policy->cBucket,size);
311         newCodePt        = policer->initialCodePt;
```

```
312         newPktType      = P_Type1;
313         break;
314       }
315       break;
316     }
317     break;
318
319   case 3:
320     switch (FlowType) {
321     case F_SHORT:
322       flow->type      = F_SHORT;
323       policy->cBucket = Ajusta(policy->cBucket,size);
324       newCodePt       = policer->initialCodePt;
325       newPktType      = P_Type0;
326       break;
327
328     case F_LONG:
329       if (( ack && policy->ack ) || (rtx && policy->rtx)) { // ACK's & RTX'x
              promotion
330         newCodePt  = policer->initialCodePt;
331         newPktType = P_Type0;
332         break;
333       }
334       if (flow->type == F_vLONG) {
335         newCodePt  = policer->downgrade2;
336         newPktType = P_Type3;
337         break;
338       }
339       flow->type = F_LONG;
340       switch ( delta ) {
341       case 0: // NO es un elefante
342         switch ( policy->BucketOpen ) {
343         case 0: // Token bucket CERRADO
344           newCodePt  = policer->downgrade1;
345           newPktType = P_Type2;
346           break;
347         case 1: // Token bucket BIERTO
348           policy->cBucket = Ajusta(policy->cBucket,size);
349           newCodePt       = policer->initialCodePt;
350           newPktType      = P_Type1;
351           break;
352         }
353         break;
354       case 1: // SI es elefante
355         flow->type = F_vLONG;
356         newCodePt  = policer->downgrade2;
```

```
357           newPktType = P_Type3;
358           break;
359         }
360         break;
361       }
362     }
363
364     PrintLine(policy->trace,(int)now, flow->type, flow->fid, pktId,
365           newPktType,ack,rtx,size,(int)policy->cBucket, flow->pkts,
366           policy->Qn,policy->flowsInst, policy->flowsAvg, newCodePt,
367           BucketLevel, policy->BucketOpen,(int)bucketTotalTime);
368
369     return (newCodePt);
370 }
371
372 /*----------------------------------------
373   double LFPPolicy::Ajusta(double bucket, double size)
374 ----------------------------------------*/
375 double LFPPolicy::Ajusta(double bucket, double size)
376 {
377   //  bucket -= size;
378
379   if ( bucket-- < 0 )
380     bucket = 0;;
381
382   //  bucket = ((bucket < 0) ? 0.00000001 : bucket);
383   return (bucket);
384 }
385
386 /*----------------------------------------
387 ----------------------------------------*/
388 void LFPPolicy::PrintLine(int trace, int now, int ftype, int fid, int pkt_id,\
389                           int pkt_type, int ack, int rtx, int size, int cBucket, int
                              pkts,\
390                           int Qn, int flowsInst, int flowsAvg, int newCodePt,\
391                           int bucketLevel, int bucketOpen, int bucketTotalTime)
392 {
393   if (trace) {
394     printf("\nP %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d",\
395           now,ftype,fid,pkt_id,pkt_type,ack,rtx,size,cBucket,pkts,Qn,flowsInst,\
396           flowsAvg,newCodePt,bucketLevel,bucketOpen,bucketTotalTime);
397   }
398   return ;
399 }
400 // End of LFP
401 ////////////////////////////////////
```

Acronyms

AF Assured Forwarding

AQM Active Queue Management

CIR Committed Information Rate

CBS Committed Burst Size

DiffServ Differentiated Services

EF Expedited Forwarding

ERED Effective RED

ETSI European Telecommunications Standards Institute

FIFO First In First Out

FSM Finite State Machine

HTTP Hypertext Transfer Protocol

IP Internet Protocol

ISP Internet Service Provider

IETF Internet Engineering Task Force

ITU International Telecommunication Union

LAS Least Attained Service

LFP Long Flow Promotions

LF Long flows

MPLS Multiprotocol Label Switching

NGN Next Generation Networks

NS2 Network Simulator 2

OSI Open System Interconnection

PDD Proportional Delay Differentiation

PDU Protocol Data Unit

PHB Per-hop behaviour

PQ Priority queuing

QoS Quality of Service

Q-SAPI Stable Queue-Based Adaptive Proportional-Integral

RED Random Early Detection

REM Random Exponential Marking

RIO RED with IN and OUT drop probability

RSVP Resource Reservation Protocol

RTT Round Trip Time

SDL Specification and Description Language

SFD Short Flow Differentiation

SHRED SHort-Lived flow friendly RED

SLA Service Level Agreement

SF Short flows

SRED Subsidised RED

SRPT Shortest Remaining Processing Time

TCP Transport Control Protocol

www World Wide Web

Bibliography

[1] B. Abbasov and S. Korukoglu. Effective RED: An algorithm to improve RED's performance by reducing packet loss rate. *J. Netw. Comput. Appl.*, 32(3):703–709, 2009.

[2] N.U. Ahmed, Q. Wang, and L. Orozco Barbosa. Systems approach to modeling the token bucket algorithm in computer networks. *Mathematical Problems in Engineering*, 8(3):265–279, 2002.

[3] A. M. Alkharasani and M. Othman. M2I2tswTCM: a new efficient optimization marker algorithm to improve fairness bandwidth in DiffServ networks. *J. Netw. Comput. Appl.*, 35(4):1361–1366, July 2012.

[4] E. Altman and T. Jimenez. Simulation analysis of RED with short lived TCP connections. *Comput. Netw.*, 44(5):631–641, 2004.

[5] G. Armitage. *Quality of service in IP networks: foundations for a multi-service Internet*. Macmillan Publishing Co., Inc., Indianapolis, USA, 2000.

[6] C. Aurrecoechea, A. T. Campbell, and L. Hauw. A survey of QoS architectures. *Multimedia Syst.*, 6(3):138–151, 1998.

[7] K. Avrachenkovt, U. Ayesta, P. Brown, and E. Nyberg. Differentiation between short and long TCP flows: predictability of the response time. In *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 762–773, March 2004.

[8] Z. Avramova, D. De Vleeschauwer, S. Wittevrongel, and H. Bruneel. Dimensioning DropTail and AQM (RED) buffers at access networks for optimal performance with bulk data TCP traffic. *Comput. Commun.*, 33:58–70, 2010.

[9] M. A. Bauer and H. A. Akhand. Managing quality-of-service in internet applications using differentiated services. *J. Netw. Syst. Manage.*, 10(1):39–62, March 2002.

[10] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol - HTTP/1.0. RFC 2616, Internet Engineering Task Force, United States, 1996.

[11] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated service. RFC 2475, Internet Engineering Task Force, United States, December 1998.

[12] T. Bonald, M. May, and J.C. Bolot. Analytic evaluation of RED performance. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1415–1424, March 2000.

[13] C. Brandauer, G. Iannaccone, C. Diot, T. Ziegler, S. Fdida, and M. May. Comparison of tail drop and active queue management performance for bulk-data and web-like Internet traffic. In *Proceedings of Sixth IEEE Symposium on Computers and Communications*, pages 122 –129, 2001.

[14] N. Brownlee and K.C. Claffy. Understanding Internet traffic streams: dragonflies and tortoises. *Communications Magazine, IEEE*, 40(10):110–117, Oct 2002.

[15] J. Cai, S.Z. Yu, and Y. Wang. The community analysis of user behaviors network for web traffic. *JSW*, 6(11):2217–2224, 2011.

[16] A. T. Campbell and G. Coulson. Implementation and evaluation of the QoS-A transport system. In *Protocols for High-Speed Networks*, volume 73 of *IFIP Conference Proceedings*, pages 201–218. Chapman & Hall, 1996.

[17] J. Cao, W.S. Cleveland, J. Gao, K. Jeffay, F.D. Smith, and M. Weigle. Stochastic models for generating synthetic HTTP source traffic. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1546–1557, March 2004.

[18] N. Cardwell, S. Savage, and T. Anderson. Modeling TCP latency. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1742 –1751, March 2000.

[19] Y. Chait, C. V. Hollot, V. Misra, D. Towsley, H. Zhang, and Y. Cui. Throughput differentiation using coloring at the network edge and preferential marking at the core. *IEEE/ACM Trans. Netw.*, 13(4):743–754, 2005.

[20] X. Chen and J. Heidemann. Preferential treatment for short flows to reduce web latency. *Comput. Netw.*, 41(6):779–794, 2003.

[21] M. Christiansen, K. Jeffay, Ott. D., and F. Donelson. Tuning RED for web traffic. *IEEE/ACM Trans. Netw.*, 9(3):249–264, 2001.

[22] K.C. Claffy, H.W. Braun, and G.C. Polyzos. A parameterizable methodology for Internet traffic flow profiling. *IEEE Journal on Selected Areas in Communications*, 13(8):1481–1494, Oct 1995.

[23] M. Claypool, R. Kinicki, and M. Hartling. Active queue management for web traffic. In *IEEE International Conference on Performance, Computing and Communications*, pages 531–538, 2004.

[24] W.S. Cleveland and D.X. Sun. Internet traffic data. *Journal of the American Statistical Association*, 95:979–985, 2000.

[25] M. E. Crovella, M.S. Taqqu, and A. Bestavros. *Heavy-tailed probability distributions in the World Wide Web*, pages 3–25. Birkhauser Boston Inc., Cambridge, MA, USA, 1998.

[26] M.E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM Trans. Netw.*, 5(6):835–846, 1997.

[27] B. Davie, A. Charny, K. Benson, J. Le, W. Courtney, S. Davari, and V. Firoiu. An expedited forwarding PHB (Per-Hop behavior). RFC 3246, Internet Engineering Task Force, March 2002.

[28] V. Deart, V. Mankov, and A. Pilugin. HTTP traffic measurements on access networks, analysis of results and simulation. In *Proceedings of the 9th International Conference on Smart Spaces and Next Generation Wired/Wireless Networking and Second Conference on Smart Spaces*, NEW2AN '09 and ruSMART '09, pages 180–190, Berlin, Heidelberg, 2009. Springer-Verlag.

[29] S. Deb, A. Ganesh, and P. Key. Resource allocation between persistent and transient flows. *IEEE/ACM Trans. Netw.*, 13:302–315, April 2005.

[30] T. Demoor, J. Walraevens, D. Fiems, and H. Bruneel. Performance analysis of a priority queue: Expedited forwarding PHB in DiffServ. *AEU-Int. J electron. C.*, 65(3):190–197, 2011.

[31] M. A. Elshaikh, M. Othman, S. Shamala, and J. M. Desa. A new fair marker algorithm for DiffServ networks. *Comput. Commun.*, 31(14):3064–3070, 2008.

[32] W. Fang and L. Peterson. Inter-AS traffic patterns and their implications. In *Global Telecommunications Conference. GLOBECOM'99*, volume 3, pages 1859–1868, 1999.

[33] L. Farmer and K.S. Kim. Cooperative ISP traffic shaping schemes in broadband shared access networks. In *4th International Workshop on Fiber Optics in Access Network, FOAN 2013*, pages 21–25, 2013.

[34] P. Ferguson and G.f Huston. *Quality of service: delivering QoS on the Internet and in corporate networks.* John Wiley & Sons, Inc., New York, USA, 1998.

[35] V. Firoiu, J.Y. Le Boudec, D. Towsley, and Z.L. Zhangm. Theories and models for Internet quality of service. *Proceedings of the IEEE*, 90(9):1565–1591, Sep 2002.

[36] P. Florissi. Quality of service management automation in integrated distributed systems. In *Proceedings of the 1994 Conference of the Centre for Advanced Studies on Collaborative Research*, CASCON'94. IBM Press, 1994.

[37] S. Floyd, R. Gummadi, and S. Shenker. Adaptive RED: an algorithm for increasing the robustness of RED's active queue management. Technical report, International Computer Science Institute, August 2001.

[38] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.*, 1(4):397–413, 1993.

[39] M. Fry, V. Witana, P. Ray, and A. Seneviratne. Managing QoS in multimedia services. *J. Network Syst. Manage.*, 5(3):283–300, 1997.

[40] O. Gandouet and A. Jean-Marie. Loglog counting for the estimation of IP traffic. In *Proc. of the Fourth Colloquium on Mathematics and Computer Science Algorithms, Trees, Combinatorics and Probabilities*, pages 119–128, Nancy (France), Sept 2006.

[41] D. L. Goldsmith, B. Liebowitz, K. Park, S. Wang, B. Doshi, and J. Kantonides. Precedence and quality of service (QoS) handling in IP packet networks. In *Proceedings of the 2006 IEEE Conference on Military Communications*, MILCOM'06, pages 666–671, Piscataway, NJ, USA, 2006. IEEE Press.

[42] R. Guérin and V. Peris. QoS in packet networks: basic mechanisms and directions. *Comput. Netw.*, 31(3):169–189, 1999.

[43] L. Guo and I. Matta. The war between mice and elephants. In *Ninth International Conference on Network Protocols, ICNP 2001*, pages 180–188. IEEE Computer Society, Nov. 2001.

[44] L. Guo and I. Matta. Scheduling flows with unknown sizes: approximate analysis. In *Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 276–277, New York, NY, USA, 2002. ACM Press.

[45] X. Guo, Z. Shan, and C. Wang. Research on web QoS control strategy based on user behaviour. In *Web-Age Information Management, 2008. WAIM '08.*, pages 564 –568, July 2008.

[46] M. Harchol-Balter, B. Schroeder, N. Bansal, and M. Agrawal. Size-based scheduling to improve web performance. *ACM Trans. Comput. Syst.*, 21(2):207–233, 2003.

[47] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured forwarding PHB group. RFC 2597, Internet Engineering Task Force, June 1999.

[48] J. Heinanen and R. Guerin. A single rate three color marker. Technical Report 2697, Internet Engineering Task Force, September 1999.

[49] J. Heinanen and R. Guerin. A two rate three color marker. RFC 2698, Internet Engineering Task Force, 1999.

[50] E. Huizingh. The content and design of web sites: an empirical study. *Inf. Manage.*, 37:123–134, April 2000.

[51] K. Hyunchul, KC. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee. Internet traffic classification demystified: myths, caveats, and the best practices. In *CONEXT'08: Proceedings of the ACM CoNEXT Conference*, pages 1–12, New York, NY, USA, 2008.

[52] R. Idris, E. Biersack, and G. Urvoy-Keller. Size-based scheduling to improve the performance of short TCP flows. *Network, IEEE*, 19(1):12–17, 2005.

[53] V. Jacobson, K. Nichols, and K. Poduri. An expedited forwarding PHB. RFC 2598, Internet Engineering Task Force, June 1999.

[54] A. Kantawala and J. Turner. Queue management for short-lived TCP flows in backbone routers. In *Global Telecommunications Conference. GLOBECOM'02. IEEE*, volume 3, pages 2380–2384, Nov. 2002.

[55] K. Lakshman and Raj Yavatkar. AQUA: A adaptive end-system quality of service architecture. In *High-Speed Networking for Multimedia Applications*, pages 155–177. Kluwer, 1995.

[56] K. Lan and J. Heidemann. A measurement study of correlations of Internet flow characteristics. *Comput. Netw.*, 50(1):46–62, 2006.

[57] L. Le, J. Aikat, K. Jeffay, and F. Smith. The effects of active queue management on web performance. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM'03, pages 265–276, New York, NY, USA, 2003. ACM.

[58] L.. Leonard and C. Riemenschneider. What factors influence the individual impact of the web? An initial model. *Electron. Market.*, 18(1):75–90, 2008.

[59] M. Looney and O. Gough. A provision aware proportional fair sharing three colour marker. *J. Netw. Comput. Appl.*, 36(1):476–483, January 2013.

[60] Hui-Lan Lu and I. Faynberg. An architectural framework for support of quality of service in packet networks. *Comm. Mag.*, 41(6):98–105, June 2003.

[61] T. Maertens, J. Walraevens, and H. Bruneel. Priority queueing systems: from probability generating functions to tail probabilities. *Queueing Syst. Theory Appl.*, 55:27–39, January 2007.

[62] B. Mandelbrot. *The Fractal Geometry of Nature*. August 1982.

[63] R. Marie, M. Blackledge, and H. Bez. Characterization of internet traffic using a fractal model. In *Proceedings of the Fourth IASTED International Conference on Signal Processing, Pattern Recognition, and Applications*, SPPRA'07, pages 253–258, Anaheim, CA, USA, 2007.

[64] M. Mellia, I. Stoica, and H. Zhang. TCP model for short lived flows. *IEEE Communications Letters*, 6:85–87, 2002.

[65] M. Mellia, I. Toica, and Hui Z. Packet marking for web traffic in networks with RIO routers. In *Global Telecommunications Conference. GLOBECOM'01.*, volume 3, pages 1828–1833, 2001.

[66] J. Miller and T. Huynh. Investigating the distributional property of the session workload. *J. Web Eng.*, 9:25–47, March 2010.

[67] S. Mo and K. Chung. A fair bandwidth distribution mechanism in a Diffserv network. *Comput. Commun.*, 30(2):358–368, January 2007.

[68] T. Mori, M. Uchida, R. Kawahara, J. Pan, and S. Goto. Identifying elephant flows through periodically sampled packets. In *IMC'04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 115–120, New York, NY, USA, 2004.

[69] D. Mustill and P. J. Willis. Delivering QoS in the next generation network - a standards perspective. *BT Technology Journal*, 23(2):48–60, April 2005.

[70] R.i Nasser-Eddine. Mathematical framework towards the analysis of a generic traffic marker. *Int. J. Commun. Syst.*, 20(4):461–475, 2007.

[71] The network simulator NS2. `http://www.isi.edu/nsnam/ns/`.

[72] D. Olshefski and J. Nieh. Understanding the management of client perceived response time. *SIGMETRICS Perform. Eval. Rev.*, 34(1):240–251, June 2006.

[73] G. Panza, S. Grilli, E. Piri, and J. Vehkapera. Relative QoS provisioning over next-generation networks. pages 160–167, 2013.

[74] K. Papagiannaki, N. Taft, S. Bhattacharyya, P. Thiran, K. Salamatian, and C. Diot. A pragmatic definition of elephants in Internet backbone traffic. In *IMW'02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement*, pages 175–176, New York, NY, USA, 2002.

[75] Eun-Chan Park and Chong-Ho Choi. Adaptive token bucket algorithm for fair bandwidth allocation in DiffServ networks. In *Global Telecommunications Conference, 2003. GLOBECOM'03. IEEE*, volume 6, pages 3176–3180, Dec. 2003.

[76] C. Partridge. A proposed flow specification. RFC 1363, Internet Engineering Task Force, September 1992.

[77] V. Paxson and S. Floyd. Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Trans. Netw.*, 3(3):226–244, Jun. 1995.

[78] P. Perry and T. Tai. Network traffic characterization using token bucket model. In *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 1, pages 51–62, March 1999.

[79] L. Popa, A. Ghodsi, and I. Stoica. HTTP as the narrow waist of the future Internet. In *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, volume 6, pages 1–6, New York, USA, 2010.

[80] X. Qi and B. Davison. Web page classification: Features and algorithms. *ACM Comput. Surv.*, 41(12):1–31, February 2009.

[81] L. Quan and J. Heidemann. On the characteristics and reasons of long-lived Internet flows. In *Proceedings of the 10th annual conference on Internet measurement*, IMC'10, pages 444–450, New York, NY, USA, 2010.

[82] D. Rossi, C. Casetti, and M. Mellia. A simulation study of web traffic over DiffServ networks. In *Global Telecommunications Conference. GLOBECOM'02.*, volume 3, pages 2578–2582, Nov. 2002.

[83] K. J. Sanjay and H. Mahbub. *Engineering Internet QoS*. Artech House, Inc., Norwood, MA, USA, 2002.

[84] M. Shifrin and I. Keslassy. Small-buffer networks. *Comput. Netw.*, 53(14):2552–2565, September 2009.

[85] L. Shiyin, X. Dong, and Q. Jiansheng. TCP flow traffic self-similar analysis and modeling. In *Control and Decision Conference. CCDC 2008*, pages 378–382, July 2008.

[86] V. Siris and H.. Marinakis. Adaptive packet marking for achieving fairness in DiffServ networks. *Comput. Commun.*, 29(1):52–58, 2005.

[87] W. Stallings. *High Speed Networks and Internets: Performance and Quality of Service*. Prentice Hall PTR, 2001.

[88] R. Stankiewicz and A. Jajszczyk. New mathematical models for token bucket based meter/markers. *Autonomic Principles of IP Operations and Management*, 4268:96–107, 2006.

[89] H. Su and M. Atiquzzaman. Comprehensive performance model of differentiated service with token bucket marker. *IEE Proceedings on Communications*, 150(5):347–53, Oct. 2003.

[90] H. Su and M. Atiquzzaman. Itswtcm: a new aggregate marker to improve fairness in diffserv. *Comp. Comm.*, 26(9):1018–1027, 2003.

[91] P. Subharthi, P. Jianli, and R. Jain. Architectures for the future networks and the next generation Internet: a survey. *Comput. Commun.*, 34(1):2–42, January 2011.

[92] S. Sudha and N. Ammasaigounden. An aggregate marker for bandwidth fairness in DiffServ. *J. Netw. Comput. Appl.*, 35(6):1973–1978, November 2012.

[93] I. Sunghwan and S. P. Vivek. Towards understanding modern web traffic. In *Proceedings of the ACM SIGMETRICS joint International Conference on Measurement and Modeling of Computer Systems*, pages 143–144, New York, USA, 2011.

[94] K. Thompson, G.J. Miller, and R. Wilder. Wide-area Internet traffic patterns and characteristics. *Network, IEEE*, 11(6):10–23, Nov. 1997.

[95] S. Tompros, D. Vergados, N. Mouratidis, and S. Denazis. NGN networks: A new enabling technology or just a network integration solution? In *Proceedings of the 5th International ICST Mobile Multimedia Communications Conference*, volume 5 of *Mobimedia'09*, pages 1–5, 2009.

[96] D. Towsley. Providing quality of service packet switched networks. In *Performance Evaluation of Computer and Communication Systems, Joint Tutorial Papers of Performance '93 and Sigmetrics '93*, pages 560–586, London, UK, 1993.

[97] E. Tsolakou, E. Nikolouzou, S. Maniatis, and I. Venieris. Definition and performance evaluation of network services deployed over a differentiated services network. *J. Netw. Syst. Manage.*, 12:537–565, December 2004.

[98] A. Vishwanath, V. Sivaraman, and M. Thottan. Perspectives on router buffer sizing: recent results and open problems. *SIGCOMM Comput. Commun. Rev.*, 39:34–39, March 2009.

[99] F. Wamser, R. Pries, D. Staehle, K. Heck, and P. Tran-Gia. Traffic characterization of a residential wireless Internet access. *Telecomm. Systems*, 48:5–17, 2010.

[100] B. Wang, B. Kasthurirangan, and J. Xu. Subsidized RED: an active queue management mechanism for short-lived flows. *Comput. Commun.*, 28(5):540–549, 2005.

[101] Z. Wang. *Internet QoS: architectures and mechanisms for Quality of Service*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[102] M. Welzl. *Network Congestion Control: Managing Internet Traffic*. John Wiley & Sons, 2005.

[103] Y. Xiang, W. Lei, and S. Huang. Some methods to improve efficiency of DiffServ. In *IEEE International Conference on Communications, Circuits and Systems and West Sino Expositions*, volume 1, pages 685–689, June 2002.

[104] X. Xiao and L.M. Ni. Internet QoS: a big picture. *Network, IEEE*, 13(2):8–18, March 1999.

[105] X. Yang. Designing traffic profiles for bursty Internet traffic. In *Global Telecommunications Conference. GLOBECOM '02. IEEE*, volume 3, pages 2149–2154, Nov. 2002.

[106] Z. Yang, S. Cai, Z. Zhou, and N. Zhou. Development and validation of an instrument to measure user perceived service quality of information presenting web portals. *Inf. Manage.*, 42(4):575–589, May 2005.

[107] I. Yeom and R. Narasimha. Marking for QoS improvement. *Comput. Commun.*, 24(1):35–50, 2001.

[108] S. Yilmaz and I. Matta. On class-based isolation of UDP, short-lived and long-lived TCP flows. In *Proceedings on Ninth International Symposium onModeling, Analysis and Simulation of Computer and Telecommunication Systems,*, pages 415–422, Aug. 2001.

[109] Y. Zhang and L. Qiu. Understanding the end-to-end performance impact of red in a heterogeneous environment. Technical report, Cornell University, Ithaca, NY, USA, 2000.

[110] X. Zhou, J. Wei, and X. Xu. Quality-of-service differentiation on the Internet: a taxonomy. *J. Netw. Comput. Appl.*, 30:354–383, 2007.

[111] X. Zhu, J. Yu, and J. Doyle. Heavy tails, generalized coding, and optimal web layout. In *INFOCOM*, pages 1617–1626, 2001.