Adaptive Admission Control in a NGN Service Platform

André Ferreira[1], Paulo Carvalho[2], Solange Rito Lima[2]

[1]Portugal Telecom Inovação, SA, 3810-1/6 Aveiro, Portugal
[2]Department of Informatics, University of Minho, 4710-057 Braga, Portugal

Abstract — **In NGN service-provisioning platforms the existence of an efficient and flexible admission control mechanism is essential for providing quality of service in a reliable and stable way, avoiding congestion scenarios caused by indiscriminate and uncontrolled service requests. The capability of modulating and regulating the rate of call acceptance, and provide service differentiation allow indirect control of the load submitted to the platform. This paper presents a service admission control solution that enables to differentiate, limit and modulate the rate by which service requests are submitted into a NGN service-provisioning platform. The solution is focused on providing a fair level of bandwidth sharing among service classes, in a configurable and dynamic way so that it can adapt the distribution by which service requests are served. To sustain the design goals of our solution, major scheduling disciplines and rate control mechanisms are here studied and compared in order to elect the more adequate components. The implemented solution was submitted to unit and charge tests; the results show its effectiveness and robustness in controlling and differentiating incoming service calls.**

***Index Terms* — Admission Control, Quality of Service, Scheduling, Service Differentiation.**

I. Introduction

Within Next Generation Networks (NGN) context, the communication process between clients and service-provisioning platforms is supported by service requests and responses. The client submits its request to the platform and awaits the respective response, which is returned once the request is served. This communication is carried through interfaces that receive the requests and invoke the respective services provided by the platform.

Service-provisioning requests have an online profile requiring to be served within a short period of time. If requests are submitted into the platform without limiting and controlling its admission rate, there may be periods where large bursts of requests are accepted consecutively, as well as moments where that rate is very low. To increase admission efficiency, one could take advantage of low activity periods to admit requests that could not be accepted under scenarios of congestion.

The execution of requests inevitably consumes memory and processing resources, more or less severely depending on the type of operations necessary to fulfill the services required. This may induce high load into the platform, leading to system congestion and to the retention of too many resources. In the most critical periods, when an excessive number of requests need to be satisfied, a point of rupture may occur causing service-provisioning disruption. At such point, available resources are not sufficient to satisfy the amount of requests to process. Hence, the ability to differentiate and prioritize more critical service requests becomes crucial.

It is widely accepted that the concept of service class allows a more efficient service differentiation process by reducing the number of levels to differentiate. Thus, services that require a similar treatment can be aggregated into the same class, so that they can be treated according to a common profile.

Thus, the motivation of this paper is to present an adaptive admission control solution that allows to differentiate, limit and modulate the rate by which service requests are handled in a NGN service-provisioning platform. The solution, developed in Java2EE, was implemented and integrated as an application server into an interface of an operational service-provisioning platform, using recent traffic control approaches. As the test results illustrate, our solution is effective in increasing the service provisioning levels in multiple service classes and can be easily deployed in NGN platforms.

This paper is organized as follows: a study of relevant scheduling disciplines and rate control methods is presented in Section II; the specification of the developed solution, including its design goals and main features, is provided in Section III; relevant tests and results are presented in Section IV; and finally, the main conclusions are summarized in Section V.

II. State Of The Art

Controlling a differentiated admission of incoming service requests can be accomplished through adequate scheduling and rate control tasks. A convenient choice and articulation of these tasks is crucial to improve the global efficiency of the service-provisioning platform. The following topics present and compare some of the most relevant mechanisms to ground the design decisions of the proposed solution.

A. Scheduling Disciplines

In admission control it is important to deal with the simultaneous arrival of contending service requests to be served. A proper queuing system can reduce the problem of contention by buffering and deciding the order by which the service requests are dispatched [1]. The ability to dispatch different service requests at distinct rates allows differentiating the amount of bandwidth used by each service type over time. However, this feature requires a fair allocation of resources while guaranteeing a good performance level, and ensuring that each request is served during a finite period of time.

In this section, we survey major scheduling disciplines, taking GPS (Generalized Processor Sharing) model [2] as a reference. Some of these disciplines, here focusing on flow admission, will be later explored for pursuing a fair differentiated admission of service requests.

GPS is a theoretical scheduling discipline that serves infinitesimal flows in a continuous and parallel mode, with perfect fairness over time. This algorithm allows allocating different bandwidth levels among active flows. GPS uses a queue for each flow. For N active queues, their flows are served simultaneously, being guaranteed to each one a ratio of at least 1/N of the total bandwidth capacity. Although GPS model is a reference model of fairness, it is not implementable. It is the infinitesimal nature of this discipline, where flows are considered to be contiguous streams of data infinitely divisible that makes it impossible to implement. Implementable models serve only one queue at a time, iteratively, which makes difficult to keep fairness over time. Thus, fair scheduling disciplines try to get as close as possible to the GPS model.

RR (Round-Robin) scheduling discipline is an attempt to deal with N flows fairly, guaranteeing a rate of 1/N for each flow. When combined with a congestion control mechanism, and fixed size packets, RR is considered the simplest method for achieving fairness in scheduling [3]. However, despite the bandwidth guarantee to all active queues, this discipline is not adaptive. Traditionally, the behavior of RR consists in a static cyclic service, serving one packet per active flow. This assures the service of each active flow, being impossible the occurrence of denial of resources that could lead to starvation [4]. In addition, the service of variable size packets penalizes queues that have a higher proportion of small packets.

Disciplines based on RR usually have low time complexity, especially advantageous when computing resources are scarce. The WRR (Weighted Round Robin) discipline applies the principles of RR, adding the capability of allocating weights to each queue, for a proportional share of resources. This feature approaches WRR fairness closer to GPS than RR. WRR establishes the quantity of packets served for each active queue, uniformly and proportionally to the defined weights. However, it is necessary to know each queue average packet size, which is problematic and in short term may lead to unfairness. In addition, there is a compromise between flexibility and delay, since large weight variations increase the delay that packets suffer in worst case scenarios [2].

DRR (Deficit Round Robin) algorithm [5] uses the principles of WRR, adding the capability of handling variable size packets without considering their average size, i.e., using a counter called deficit counter. Thus, if a queue has a deficit counter value greater or equal to the size of the packet, then the packet is served, and the counter is updated according to its difference with the served packet size. This discipline is a better approximation to GPS model than RR and WRR.

Proposed in [6], the algorithm Fair Queue (FQ) is a non-infinitesimal scheduling discipline that aims to approach GPS model. FQ considers a queue per flow and the packet size, avoiding bandwidth monopolization, which may cause starvation to other flows. If a flow does not require all the bandwidth it is entitled for, the residual bandwidth is shared fairly among active flows that require it. For each packet, FQ computes a start and finish virtual time function, being the packets served according to the order of finish virtual time. The finish virtual time is the sum of the star virtual time to the GPS time of transmission of the packet. The main advantage of FQ is to protect well-behaved flows against bad-behaved ones, providing a good level of fairness. However, FQ does not assign weights in order to distinguish flows costs, not providing differentiation. This algorithm also has the disadvantage of requiring complex computation of virtual time functions at the arrival of each packet.

FQ based disciplines allowing differentiation, such as PGPS (Packet-by-Packet GPS) [2] and WFQ (Weighted Fair Queuing) [7], have been proposed. The main idea of these identical disciplines is to match the computation of each packet virtual finish time to a GPS system, being a good approximation to GPS. Besides all the advantages of FQ, they assign weights to the queues. This allows the regulation of the number of packets served by each queue. The algorithm computes a finish virtual time function for each arrived packet, which determines the service time. The virtual time function is calculated according to the bandwidth, the weight of the queue, the size of packets and an indicator that represents the number of rounds served. The round time and the packets' delay increase with the number of active sessions [2]. The method for virtual time computation enables WFQ to achieve a good approximation to GPS model. However, part of the high complexity of WFQ results from the computation of a virtual time function that uses the time identifier that each packet would have in GPS model. Compared with WRR, which also associate weights to queues, WFQ performs a more efficient management. As exemplified in [2], in the worst case, the PGPS algorithm is closer to GPS than the WRR algorithm.

In [8], it is shown that the WFQ algorithm can serve more packets than GPS. Although the WFQ can serve packets faster than the ideal GPS, it fails the supposed scheduling efficiency. W2FQ (Worst Case Fair Weight Fair Queuing) assures the same level of fairness and delay guarantees as WFQ, and it was developed to address the fact that the WFQ is not as close to GPS as expected [8]. W2FQ grants fairness even in worst-case scenarios. This is proved through the worst-case fair index, which is a metric that measures the discrepancy between a discrete iterative scheduling model and idealistic infinitesimal GPS model. The disadvantages of W2FQ are that, as the WFQ algorithm, the time complexity is high due to the iterative computation of complex virtual time functions.

In an attempt to reduce the complexity that characterizes WFQ and W2FQ algorithms, while maintaining its properties, the algorithm SCFQ (Self-Clocked Fair Queuing) was proposed in [9]. The operation of this algorithm is similar to WFQ, however, virtual finish time is computed considering a time tag related to the last served packet. The packet is then inserted into a queue and waits for service and, like WFQ, the scheduler serves packets by its finish time order. In contrast to WFQ, SCFQ has its own time reference, which measures the service progress through a virtual time function that depends exclusively on the progress of served queues [9]. SCFQ allows a fair scheduling, allocating the bandwidth efficiently between the queues. It is more easily implementable than WFQ and provides similar guarantees. However, despite the low complexity, SCFQ performs below WFQ and may be unfair in short term operation, especially when the number of flows increases [8]. In [10], it is presented an efficient implementation of SCFQ, allowing the computation of virtual time function when packets arrives to the head of the queue.

In the last few years, new scheduling disciplines have been proposed. A credit-based fair scheduling discipline called MCF (Most Credit First) was presented in [11]. Its algorithm minimizes the difference between the service that a flow should receive in an ideal fairness model and the one that is actually received. It works by assigning a credit value to each flow. The flows are served in a balanced mode, according to its credits, and considering their weights. The flow with more available credits is served iteratively. Fairness is provided restricting the value of the accumulated credit. Flows that require bandwidth and have negative available credits are penalized by not transmitting until their accumulated credit is recovered.

In [11], it is also presented the algorithm FMCF (Fast Most Credit First) that reduces the logarithmic time complexity of MCF to a constant time complexity. It is shown that these disciplines perform better than WFQ and DRR.

Other credit-based discipline, presented in [12], is CBFQ (Credit-Based Fair Queuing). An attractive feature of CBFQ is the use of different counters to track the amount of accumulated credits reflecting the bandwidth used for each flow. CBFQ considers every relevant aspect of a fair adaptive algorithm, including packet size as well as the current length and weight defined for each active queue. Based on these metrics, CBFQ decides which flow should be served iteratively, maintaining fairness over time. The service is balanced so that the expected percentage of service is maintained over time. Thus, this discipline achieves the same level of fairness and delay guarantees as virtual time approaches, avoiding their disadvantages. Compared to alternative approaches, such as SCFQ, WFQ/PGPS, this discipline also provides easier implementation [12].

MCF and CBFQ have the advantages of the algorithms which resort to virtual time functions, while avoiding their complexity and being more implementable. Therefore these algorithms constitute an appropriate choice to integrate the proposed admission control solution.

B. Rate Control

A rate control mechanism controls the pace at which requests are submitted into the platform, shaping it so that the rate is limited to the expected granularity. A simple method to limit the rate is based on the *Leaky Bucket*. As known, although this algorithm can effectively limit the rate at which requests are sent, it is inefficient in cases where the rate limit is rarely reached. If no information is received during a certain period of time, the unused bandwidth cannot be used for future transmissions, leading to low service utilization. The common *Token Bucket* method provides a more flexible rate regulation by allowing the admission of bursts of requests. Usually, the admission of each request consumes a token. The admission rate is therefore determined by the rate at which tokens are added to the bucket. Due to its properties, a *Token Bucket* based algorithm is here adopted for modulating the rate of admission control.

III. Proposed Admission Control Solution

In this section, it is presented an adaptive admission control solution, developed for a real-time NGN service-provisioning platform. The solution was developed in Java2EE, and integrated in an interface of the service platform.

A. Design Goals

The proposed admission control solution considers the following design goals:

- *limitation of the requests admission rate*; when exceeded, it is possible to store temporarily candidate requests, preventing a direct discard;
- *fair service differentiation,* according to the priority defined for each service class;
- *adaptive behavior,* according to the incoming load, providing flexible bandwidth sharing over time;
- *fully configurable*, so that the expected rate control granularity and differentiation behavior can be obtained;
- *low-time complexity*, without compromising efficiency.

B. Relevant Features

The proposed solution is responsible for maintaining the levels of service quality, supporting differentiated service requests waiting for admission. In this context, the scheduling discipline is a fundamental admission control element. To be effective and fair [13] its main characteristics are as follows:

1. *Low Complexity and Efficiency:* scheduling should be computationally simple, while maintaining efficiency. The iterative process of decision should have low computational complexity, without disregarding the initial objectives for the scheduling discipline. The complexity should be O(1) so that the time taken to select the service request to serve for each iteration does not depend on the number of requests.

2. *Scalability*: the algorithm must be scalable, therefore it must have a good temporal computational complexity so that the scheduling process can be efficient both in small and large scale.

3. *Fairness*: it is essential to guarantee fairness in the scheduling process. Therefore, enough resources need to be allocated to each queue to ensure that no queue remains indefinitely without being served. To prevent misuse of bandwidth and to avoid starvation it is essential to ensure that scheduling of a particular class of service does not degrade the service of the others.

4. *Adaptation*: Sometimes it is impossible to guarantee the scheduling of all requests within a short period of time. Therefore, scheduling must be optimized in order to minimize the number of requests discarded. The algorithm must react to the occurrence of situations that may affect the quality of service. Every time that the load of each queue or the admission pattern changes dramatically, the scheduler should adapt its behavior to maximize the acceptance rate.

5. *Differentiation*: The scheduling algorithm must have the capacity of service differentiation according to distinct types of service classes. The requests related to each service class should be placed in the same queue so that the expected behavior can be provided, depending on the settings specified in the service level agreements. The value of each priority must be considered in order to avoid introducing too much latency in the admission of low priority queues.

6. *Quality of Service Guarantees*: The performance levels should be defined in contracts between the customer and the service provider. The degradation of QoS should be avoided. The control of metrics such as bandwidth, delay, loss, etc. can ensure the performance of the admission mechanism, according to the level of QoS required for each class.

In operational NGN service platforms, it is advisable to limit the time a request waits for admission control so that when a request times out, it should be removed from the queue. The configuration of these parameters should be carefully set, since they affect the overall performance of admission control.

C. *Integration Environment*

As mentioned, the admission control strategy is integrated in the access interface of a service-provisioning platform, which handles *HTTP URL Encoded* service requests submitted by external entities. This access interface developed in Java2EE is executed in platforms supporting application servers *WebLogic* and *JBoss*.
As illustrated on Figure 1, the access interface is divided in two operational levels: (i) the interface level, establishing the connection to external entities; (ii) the engine, providing the functional logic and rules for accessing services.

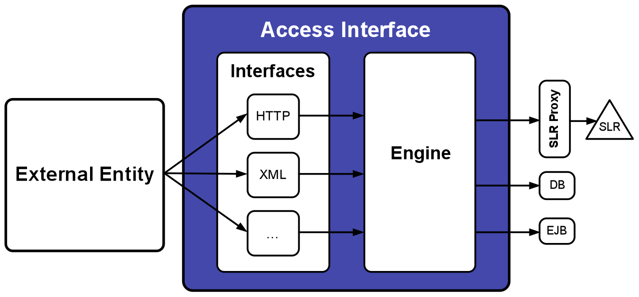


Figura 1. Simplified architecture of the Access Interface

Service requests submitted by external entities to the access interface undergo a validation and admission process before being forward to the engine. First, service requests receive an identifier from the corresponding interface. Next, the engine verifies if the external entity has permission to access the requested service. It also verifies if the service request time is within the agreed time schedule and if the maximum number of requests negotiated with the external entity is not being exceeded. If the service request is valid, the invoked service is executed; this service may correspond to an invocation of primitives, of database functions or procedures, or of EJBs (Enterprise JavaBeans) methods. The service response to the external entity is provided in XML format. When the service is denied, a reporting message is also provided.
External entities submit service requests at an unpredicted rate. Thus, when forwarding a service request from the interface to the engine, the admission process is also responsible for limiting, differentiating and modulating the rate of service requests. This admission process is detailed below.

D. *Admission Process Stages*

The proposed admission process is applied to every request arriving to the access interface. As illustrated in Figure 2, the admission of a service request undergoes several stages.

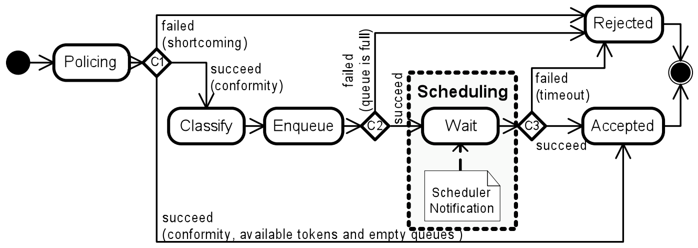
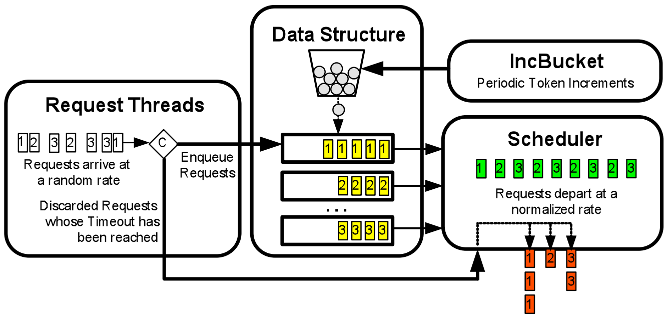


Figure 2. Admission Process

Policing: This component verifies if the request has permission to access the platform, according to the corresponding contract. It is confirmed if the client is allowed to invoke the specified service, and if the service can be executed at the current time, according to the negotiated time schedule. If the request passes the policing process (condition C1) it proceeds to classification, otherwise it is rejected.
Classification: Similar priority services are grouped in the same service class. Classification is the stage that maps the service request to an existing service class. This information is stored in a database table, being kept in memory when needed. This allows the classification process to be done quickly. After being classified, the request is inserted into the respective queue, and waits for the scheduler to serve it. If the request identifier is successfully placed into the queue (condition C2), it is automatically submitted to the scheduling process, which runs in parallel. If the queue is full, the request is rejected. The use of service classes reduces the number of queues used in scheduling, by avoiding the use of a queue per service.
Scheduling: The scheduler serves the active queues iteratively, controlling the order by which requests are served. Assuming that an efficient fair scheduler must support the features presented in Subsection III-*B*, the most appropriate scheduling disciplines are MCF and CBFQ. Other disciplines are inadequate, either because they perform the computation of virtual time functions per request, or because they do not lead to a suitable level of fairness. Thus, the proposed solution implements credit-based scheduling algorithms MCF, to achieve a fair and efficient scheduling, and CBFQ, to achieve a fair but more adaptive solution, when considering load conditions and the utilization level of each queue over time.

E. *Admission Process Architecture*

The admission process is implemented according to the architecture illustrated in Figure 3.


Data Structure
IncBucket
Periodic Token Increments
Request Threads
Requests arrive at a random rate
Discarded Requests whose Timeout has been reached
C
Enqueue Requests
Scheduler
Requests depart at a normalized rate
Figure 3. Architecture Scheme

Service-provisioning requests are received by the platform at a random rate, being admitted at a normalized rate. This service rate modulation is performed resorting to the following modules:

1) *Data Structures:* The data structure necessary to support admission control consists of a bucket and a queue per service class. Following *Token Bucket* terminology, the bucket is a counter that defines the granularity of requests being schedule. The periodicity of bucket refills determines the elasticity and accuracy of rate control. When the bucket is empty, it means

that the request admission rate has been reached, so the scheduler will wait for new tokens to be added to the bucket.

The data structure Queues allows storing the identifiers of requests. All requests in a queue belong to the same service class, have the same priority, and are served according to a FIFO discipline. Once inserted in the queue, the requests' thread (see below) waits for an acceptance or rejection notification. Methods of early congestion detection are not used because requests have an online profile, requiring an urgent and necessary answer. The requests cannot be considered invalid unless the timeout is exceeded.

2) Request Threads: Each request is associated with a thread. After policing the request, the request thread classifies and inserts the request into the corresponding queue. When all the queues are inactive and the admission rate is not exceeded, requests are served directly, avoiding enqueuing and scheduling (condition C). This feature increases the flexibility and efficiency of rate regulation. In these circumstances, requests are only inserted into the queue when the bucket has no more tokens available.

However, if any queue has requests to serve, they have priority, and direct admission is not allowed. Once the request identifier is inserted into the queue, threads await a service admission notification. If this notification does not arrive within a certain amount of time, a timeout occurs and the request identifier will be removed from the queue.

3) *IncBucket:* The *IncBucket* is a thread responsible for the periodic refill of the bucket. The bucket size (counter) limits the maximum burst of requests that can be sent consecutively. The bucket refill rate determines the number of units increased per iteration. Tokens may be increased in several units at once, resulting in larger periods of time between increases. This decision must be made wisely since it impacts on rate control granularity. The *IncBucket* thread operates alongside the admission mechanism.

4) Scheduler: Queues are served when there are credit units to be consumed in the bucket. For each request identifier that is served, the bucket is decremented by one unit. Scheduling of queues is carried out according to the implemented scheduling discipline (MCF or CBFQ). The service as a whole, considering all the queues, has a normalized departure rate, determined by the configured service rate.

5) *Coordination:* The presented tasks are executed in parallel, which requires coordination among threads. There are three coordination cases: (i) when the queues are empty the scheduler stays idle until a new request arrives, and a notification is sent from the request thread to the scheduler so that it can proceed; (ii) when the bucket is empty, the scheduler is also idle until the counter units is increased, and a notification is sent from *IncBucket* to the *Scheduler*, allowing the service to proceed; (iii) when the scheduler serves a request, it sends a notification to the respective thread, which is idle while waiting for a request acceptance decision.

IV. Test And Results

In order to demonstrate the effectiveness of the proposed solution, two types of load tests were carried out, through the concurrent submission of service requests to the system access interface.

Three distinct service classes were defined and associated with a corresponding service queue per class, with priorities 0.5, 0.3 and 0.2, respectively. For test purposes, we have defined and parameterized also three services within the NGN platform. These services just invoke a PL/SQL procedure responsible for registering the service class identifier and the timestamp in which the request is effectively served. The collected data is stored in a database for performance analysis and reporting purposes. For the test scenarios considered, the admission process was configured to limit the admission rate to 100 requests per second.

A monitoring thread was created to periodically measure the performance of the three service classes. This thread maintains counters that reflect the number of served requests for each service type, distinguishing: the number of requests served directly, without being queued; the number of requests served indirectly, through enqueuing and scheduling; and the number of requests discarded due to timeout. Thus, monitoring allows verifying that both scheduling and rate limitation perform correctly over time.

Test1: Differentiation

The performed test considers the consecutive submission of 1200 requests for class-1. After four seconds, 1200 requests are submitted for class-2, and finally, 1200 requests for class-3. As the maximum service rate is limited to 100 requests per second, in the moment that requests from class-2 and 3 are submitted, there are still requests from class 1 in the corresponding queue to be served. This allows verifying if the bandwidth sharing is performed in a fair mode among all active queues, for several levels of activity.

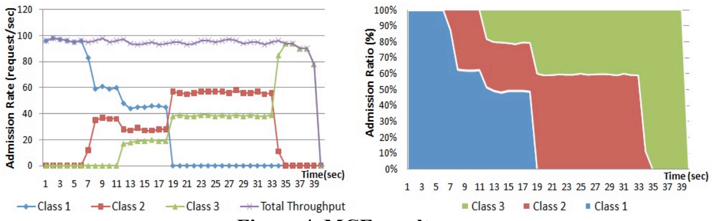


Figure 4. MCF results

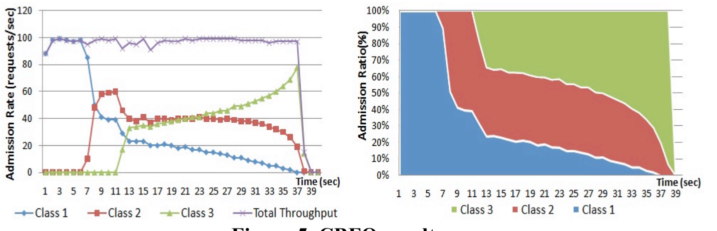


Figure 5. CBFQ results

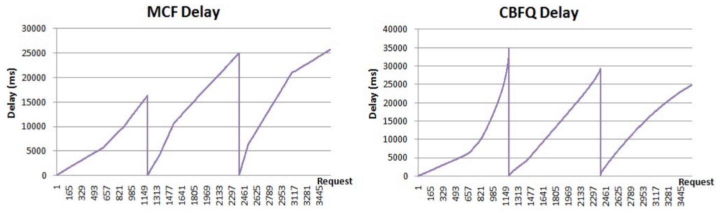


Figure 6. Service Delays

Figure 4 and Figure 5 present the test monitoring results for MCF and CBFQ algorithms respectively, regarding the achieved throughput and bandwidth occupancy. Figure 6 presents the evolution of the metric delay in both cases.

MCF results presented in Figure 4 show that, in the initial moments, when only class-1 queue is active, the entire amount of bandwidth is consumed, and a constant throughput of 100 requests per second is achieved. When class-2 requests are inserted into the respective queue, the scheduler distributes the bandwidth between active queues according to their weights. Similarly, when class-3 requests are submitted, the third queue becomes active and the bandwidth is distributed among all active queues, in a proportional mode. When class-1 queue becomes empty, its bandwidth is distributed between class-2 and class-3 queues that remain active. Finally, when class-2 queue becomes inactive, class-3 consumes the total amount of bandwidth as no other queue has competing requests. This test presents the fair behavior of MCF scheduler.

CBFQ results presented in Figure 5 show that, in the initial moments, class-1 queue also consumes the entire bandwidth, having a constant throughput of 100 requests per second, as expected. However, instead of sharing the bandwidth strictly according to queue priorities, the queue lengths are also considered. The increasing curve that characterizes class-3 service comes from the fact that, despite the lower priority associated with service class-3, the size of class-3 queue is considerable superior to class-2 queue. Thus, to balance queue lengths, class-3 queue is served with priority. This fact makes CBFQ appropriated to deal with congestion scenarios, in which lower priority queues have high traffic affluence.

MCF service delays presented in Figure 6 illustrate three variations, related to class-1, 2 and 3 requests. The delay increases because the last served packet has the higher delay. Requests from class-1 suffer lower delay because class-1 has the highest priority. Although class-3 requests have lower priority, they suffer less delay then class-2 requests. However, class-3 had the chance of using the total available bandwidth, while class-2 service was exposed to higher contention, having to share the bandwidth with the concurrent queues.

In contrast, CBFQ service delay times demonstrate that the last served request from class-3 reach the lowest values, compared to the last served requests from other classes. This is because class-3 requests were served with higher priority due to the existence of higher load in class-3 queue. Class-1 delay is the highest because while balancing the load between all service classes, the service of class-1 requests was delayed. Thus, CBFQ sacrificed worst-case delay in order to balance the load of each queue.

Test 2: Handling Bursts of Requests

In this test, we submitted a total of 900 concurrent service requests for a service class, with a rate of 110 requests per second. The test was also performed for a bucket size limited to 1, 10 and 20 tokens, respectively.

As the admission rate is limited to 100 requests per second, this test allows illustrating the system behavior in presence of bursts of requests.

In Figure 7, it is showed that the ability to serve requests directly increases for larger bucket size. Although this leads to a more relaxed admission control, it reduces queuing and scheduling computational overhead, and enhances the efficiency of our proposal in operational scenarios.

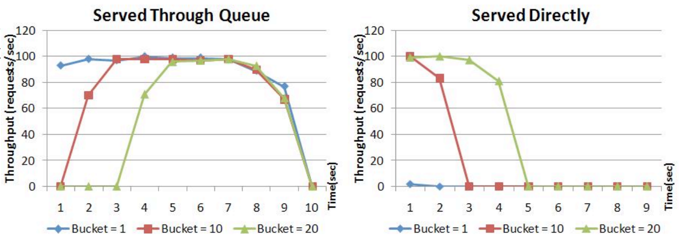


Figura 7. Handling bursts of requests

V. Conclusions

This paper has presented a differentiated admission control solution that integrates the recently proposed scheduling disciplines Most Credit First (MCF) and Credit Based Fair Queuing (CBFQ). The deployment of this solution in a NGN service-provisioning platform has allowed regulating the throughput of distinct service classes, according to each class priority, and to support the differentiation of promptness level provided among incoming services requests. This solution is a clear step towards quality of service provisioning through differentiated admission control, which is essential to enhance the performance and reliability of the NGN platform. The preliminary results have shown that the behavior of CBFQ is based on performing a fair scheduling while maintaining the length of the queues balanced. MCF performs a fair scheduling exclusively based on queues priorities, without considering the level of congestion present in the queues.

As future work we will further explore the behavior of both algorithms under distinct test scenarios and enhance the proposed admission control solution.

References

[1] Boudec, J.-Y., "*Rate adaptation, Congestion Control and Fairness: A Tutorial*". 2000.

[2] Parekh, A.K. and R.G. Gallagher, "*A generalized processor sharing approach to flow control in integrated services network: the single node case*", in IEEE/ACM Trans. on Networking *(Vol. 1)*. 1993, IEEE Press.

[3] Hahne, E.L., "*Round-Robin Scheduling for Max-Min Fairness in Data Networks*". IEEE Journal, 1991. 9: pp. 1024-1039.

[4] Trajkovic, L. *et al*, "*Modeling Packet Scheduling Algorithms in IP Routers*". 2001: School of Engineering Science, Simon Fraser University.

[5] Shreedhar, M. and G. Varghese, "*Efficient fair queueing using deficit round robin*", in SIGCOMM'95. 1995, ACM: Cambridge, Massachusetts, U.S.A.

[6] Nagle, J.B., "*On packet switches with infinite storage*, in *Innovations in Internetworking*". 1988, Artech House, Inc. pp. 136-139.

[7] Demers, A., S. Keshav, and S. Shenker, "*Analysis and simulation of a fair queueing algorithm*". ACM Computer Communications Review, 1989.

[8] Bennett, J.C.R., "*Wf2q : Worst-case Fair Weighted Fair Queueing*". 1996.

[9] Golestani, S.J., "*A self-clocked fair queueing scheme for broadband applications*", in INFOCOM'94. , 1994.

[10] Rexford, J.L., A.G. Greenberg, and F.G. Bonomi, "*Hardware-Efficient Fair Queueing Architectures for High-Speed Networks*", in INFOCOM'96. 1996. pp. 638-646.

[11] Pan D., Y.Y., "*Credit based fair scheduling for packet switched networks*", in INFOCOM'05, 2005.

[12] Bensaou, B., D.H.K. Tsang, and K.T. Chan, "*Credit-based fair queueing (CBFQ): a simple service-scheduling algorithm for packet-switched networks*". IEEE/ACM Trans. on Networking, 2001. 9(5): pp. 591-604.

[13] Briscoe, B., "*Flow rate fairness: dismantling a religion*", ACM Computer Communications Review, 2007, 37(2): p. 63-74.