Adaptive Admission Control in Real Time Systems

André Filipe dos Santos Ferreira[1,2], Solange Rito Lima [2]

[1]Portugal Telecom Inovação, SA, 3810-1/6 Aveiro, Portugal
[2]Departamento de Informática, Universidade do Minho, 4710-057 Braga, Portugal

Abstract — **In real-time 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 request admission. The capability of modeling 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 differentiated admission control solution that allows to limit and modulate the rate by which service requests are submitted into a 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 decisions of our solution, major scheduling disciplines and rate control mechanisms, some of them proposed recently, are studied and compared. The solution was submitted to unit and charge tests, whose results show its effectiveness and robustness.**

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

I. Introduction

The communication process between clients and service provisioning platforms is commonly supported by service requests and responses. The client submits its request to the platform and awaits the respective response, which is returned once the service is satisfied. 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 that raises the need of attending requests within a short period of time. If requests are submitted into the platform without limiting and controlling its rate of admission, may occur periods where large bursts of requests are accepted consecutively, as well as moments where the arrival of requests has a very low rate. This latter situation could be used to admit requests that could not be accepted under previous 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 that they claim. This may induce high levels of load into the platform, leading to system congestion and to the retention of too many resources. In the most critical periods, when a large and unsustainable number of requests need to be satisfied, a point of rupture can be reached, making the provision of services unavailable at all. At such point, available resources are not sufficient to satisfy the amount of requests to process.

Another issue that should be taken into account is the order by which applications are admitted into the platform. In this context, it is useful to give priority to more critical services. 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 service class, so that they can be treated according to its profile.

Thus, the motivation of this paper is to present a service differentiated admission control solution that allows the limitation and modulation of the rate by which service requests are submitted into a service provisioning platform.

The solution, developed in Java EE, was implemented at application level into an interface of a service provisioning platform, using recent traffic control approaches.

This paper is organized as follows: a study of relevant rate control methods, bandwidth allocation policies and scheduling disciplines is presented in Section II; the specification of the developed solution, including its design goals and main features are 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

A differentiated admission control process is accomplished through several tasks, such as rate control, bandwidth management and scheduling disciplines. It is important to select a suitable method for each one of these tasks, in order to improve the global efficiency. The following topics present and compare some of the most relevant methods sustaining the deign decisions of the proposed solution.

A. 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 desired granularity. These mechanisms usually rely on the use of queues to make temporary storage of service requests, when the limit rate is exceeded, minimizing the discard of residual requests that couldn't be attended at their arrival. The admission of such requests is carried out when the rate drops below the maximum allowed rate.

A simple method to limit the rate is based on the *Leaky Bucket*. In this method, the service requests are transmitted through a queue, called bucket. The requests arrive to the bucket at an undefined rate, and are processed at a specific rigid rate in arrival order. Requests that arrive when the queue is full are usually discarded. 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.

A more flexible method is *Token Bucket*, which has a similar philosophy to *Leaky Bucket*, supporting a more flexible rate regulation by allowing the admission of bursts of requests. This method performs admission through the consumption of

credit units, called tokens. 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. When it becomes empty the admission process idles until new tokens are added. The size of allowed bursts depends on the size of the bucket, i.e., on the maximum number of tokens that can be added to the bucket. Due to its properties, the *Token Bucket* algorithm is the strategy proposed for controlling the rate of admission control.

B. Bandwidth Management

In admission control it is important to deal with the simultaneous arrival of service requests that results in contention. A scheduling discipline can reduce the problem of contention by deciding the order as the information sources are attended. Since the scheduler cannot transmit multiple requests simultaneously, the transmission is made iteratively over existing queues.

Most of the scheduling disciplines are work-conservative. The scheduler only idles when all queues became inactive, improving bandwidth utilization over time [4].

The ability to transmit different flows 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 level of performance. It is necessary to ensure that each flow does not remain without being served during an indefinite period of time. Badly behaved flows, with large bursts, tend to consume high quantities of bandwidth, preventing the service of well behaved flows whose traffic profile does not exceed the expected rate. A common bandwidth management policy to avoid this is Max-min, which allows a fair allocation of bandwidth by distributing the available resources for all the flows in a fair mode. Only if there is bandwidth surplus from well-behaved flows, it will be shared between dissatisfied flows. Thus, the bandwidth is allocated to flows in ascending order, starting with lower rate flows, ensuring that a badly behaved flow does not exceed its allocated bandwidth nor affects well behaved flows. Thus, this process grants fairness among all flows over time. This fairness strategy is used with more or less severity by fair scheduling disciplines.

Other bandwidth allocation approach aims at optimizing flows throughput. In this perspective is assumed that each flow rate is below line capacity. The scheduler needs to know the load level of the queues and the conditions of the channel, but does not need to know the total capacity of the channel and each flow rate [5]. In [6], it is shown that resource allocation done in order to maximize the rate of admission can be a stable policy. This policy tries to maximize each flow's throughput, although it may compromise fairness between flows, which is unacceptable in most scenarios since it tends to degrade the quality of service as a whole.

Proportional Fairness [7] is focused on providing both fairness and throughput maximization. The bandwidth is distributed in order to deliver similar cost to all flows, or minimize the maximum cost. Thus, this policy aims to optimize the throughput and achieve better use of bandwidth.

C. Scheduling Disciplines

The most basic scheduling discipline is FCFS (First-Come First-Served), or FIFO (First-In First-Out), where elements are served according to their order arrival, This discipline neither support service differentiation nor provides fairness guarantees. It is normally used in situations where the sources of information can perform all the congestion treatment.

The GPS (Generalized Processor Sharing) model [17] is a theoretical scheduling discipline that serves infinitesimal flows according to Max-min criterion. The service is performed 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.

The GPS model is used as a comparative model to measure the performance of other scheduling disciplines because it is a reference model of fairness, although not implementable. It is the infinitesimal nature of this discipline, where flows are considered to be contiguous streams of information 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.

The 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 [8]. 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 [9]. However, the service of variable size packets penalizes queues that have a higher proportion of small packets.

Disciplines based in 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. The service is cyclic and static over time. For fixed size packets, several packets are served consecutively so that the weights defined for each queue are accomplished. For variable packet size, the weight is normalized according to the relation between the average packet size and the static weight. 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 [10].

A RR variant is DRR (Deficit Round Robin) algorithm [11]. It 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 [12], the algorithm Fair Queue (FQ) is a non-infinitesimal scheduling discipline that aims to approach GPS model. Unlike FCFS, where a flow can consume large amounts of bandwidth, FQ discipline serves in a proportional manner. This policy has the concept of associating a queue per flow, avoiding bandwidth monopolization, which may cause starvation to other flows.

The service is accomplished considering the packet size and distinguishing the different flows. If a flow does not require all the bandwidth he 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.

In order to cross the need of the average packet size, algorithms PGPS (Packet-by-Packet GPS) [10] and WFQ (Weighted Fair Queuing) [13] have been proposed. The disciplines are identical, but have been presented independently. The main idea of these disciplines is to match the computation of each packet virtual finish time to a GPS system. Thus, they represent a good approximation to GPS.

These disciplines have all the advantages of FQ, with the advantage of associating weights to the queues. This allows the regulation of the number of packets served by each queue [10]. 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 [10]. 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 [10], in the worst case, the PGPS algorithm is closer to GPS than the WRR algorithm.

In [14], 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. However in WF2Q (Worst Case Fair Weight Fair Queuing) is guaranteed that the amount of served information never exceeds the one served in GPS model. W2FQ 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 [14]. 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 was proposed in [15]. The operation of this algorithm is similar to the 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 [15]. 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 has performances below the WFQ and may be unfair in short term operation, especially when the number of connections increases [14]. In [2], 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. In 2005, a credit based fair scheduling discipline called Most Credit First (MCF) was presented in [16]. 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 by the algorithm. It works by associating 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 most available credit 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 [16], it is also presented the algorithm Fast Most Credit First (FMCF) that reduces the logarithmic time complexity of MCF to a constant time complexity. It is shown that these disciplines have better performance than WFQ and DRR.

Other credit-based discipline, presented in [17], is Credit-Based Fair Queuing (CBFQ). 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 actual 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 [17].

III. Proposed Admission Control Solution

In this section, it is presented an adaptive admission control solution, developed for a real-time service provisioning platform. The solution was developed in Java EE, and integrated in an interface of a service provisioning 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 temporary store rejected requests, preventing direct discard;
- *fair service differentiation,* according to the priority defined for each service class;
- *adaptive treatment,* according to the incoming load, providing flexible bandwidth sharing over time;
- *fully configurable*, so that the desired 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 request admission. The scheduling discipline is the most important admission control element that can provide it. Therefore, some desirable features must be taken into consideration, yet many of them are sometimes mutually contradictory [1]. The main characteristics of a scheduling algorithm to be effective and fair are the following:

1. *Low Complexity and Efficiency:* scheduling should be computationally simple, while maintaining efficiency. The iterative process of decision should have low complexity instructions, without disregarding the initial objectives for the scheduling discipline. The complexity should be O(1) [2, 3] so that the time taken to elect the flow to serve for each iteration does not depend on the number of flows.

2. *Scalability*: the algorithm must be scalable, therefore it must have a good temporal computational complexity so that the scheduling process can be efficient in small and large scale. This concern may be less relevant in cases where the number of queues is limited.

3. *Fairness*: it is essential to guarantee fairness in the scheduling process. It should be allocated enough bandwidth to each queue to ensure that no queue remains indefinitely without being serviced, occurring starvation. The absence of starvation implies that a minimum service allocation is established for each queue. To prevent misuse of bandwidth and to avoid starvation it is essential to ensure that the scheduling of a particular class of service does not substantially degrade the service of the others. This may be achieved protecting well-behaved flows from badly behaved flows that may endanger fairness. These disruptive flows should be ignored or penalized.

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

5. *Differentiation*: The scheduling algorithm must have the capacity of service differentiation according to the distinct types of service class. 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 service level agreements. The value of each priority must be considered in order to avoid putting 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 supplier. The degradation of quality of service should be avoided. The control of metrics such as bandwidth, delay, jitter, loss, etc. can ensure the performance of the admission mechanism, according to the level of quality of service required for each class.

In operational 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 performance of admission control.

C. Admission Process

The proposed admission process is applied to every request arriving to the interface. As illustrated in Figure 1, the admission of a service request is carried out through several stages.

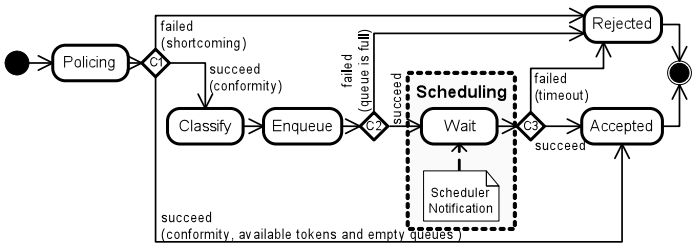

Figure 1. Admission Control Scheme

*Policing***:** it verifies if the request has permission to access the platform, according to the specified contract. It is confirmed if the client can 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, allowing controlling the order by which requests are served. An efficient fair scheduler must support the features presented in Subsection III-*B*. Thus, the most appropriate scheduling disciplines are MCF and CBFQ. All the others disciplines are inadequate, either because they perform the computation of virtual time functions per request, or because they do not achieve 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, considering load conditions and the utilization level of each queue over time.

D. System Architecture

The admission process is performed over the system architecture illustrated in Figure 2.

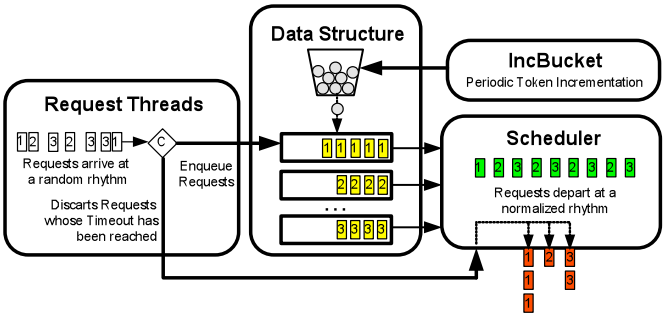

Figure 2. Architecture Scheme

Service provisioning requests are received into the platform at a random rate, departing at a normalized rate. This is performed controlling the service rate of the requests. In more detail, the architecture of the proposed system consists of four modules:

1) 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 rate is not exceeded, requests are served directly, avoiding the operation of 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 on service, and direct admission is not allowed. Once the request identifier is inserted into the queue, threads await for 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.

2) *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 number of units increased periodically in the bucket defines the rate elasticity and accuracy. 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 use of Queues allows storing the identifiers of requests. All requests in a queue belong to the same service class, have the same priority, so there is no need to sort them. Each queue is served by FIFO method. Once inserted in the queue, the requests thread waits for a notification that reflects its acceptance or rejection. 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.

3) *IncBucket:* The *IncBucket* is a thread responsible for the periodic increasing of the bucket. The bucket counter is limited by the maximum burst of requests that can be sent consecutively. The increasing rate determines the number of units increased per iteration. It is possible to increase several units simultaneously, resulting in larger periods of time between increases. This decision must be taken wisely since it sacrifices 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. The 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 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

To demonstrate the behavior of the solution several tests were performed. These tests simulate requests submitted to admission control. The solution was configured to limit the rate to 100 requests per second. The timeout was configured in order to avoid discarding requests. The bucket was initially limited to a unit to avoid bursts of requests. Consequently, it is increased each 1/(maximum rate) milliseconds. Thus, the solution is configured for a rigid rate of transmission. Three service classes were defined, supported by three queues, with a bandwidth sharing of 50%, 30% and 20%.

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. The fact that the throughput cannot overcome 100 requests per second, determines that in the moment that requests from class-2 and 3 are submitted, there are still requests from class 1 in the respective queue to be served. This allows verifying if the bandwidth sharing is performed in a fair mode among all active queues, in several cases of activity.

To evaluate the behavior of the proposed admission control scheme, 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 the scheduling and rate limitation performs correctly over time.

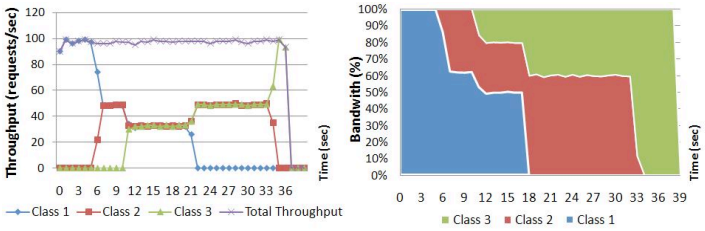

Figure 3. MCF results

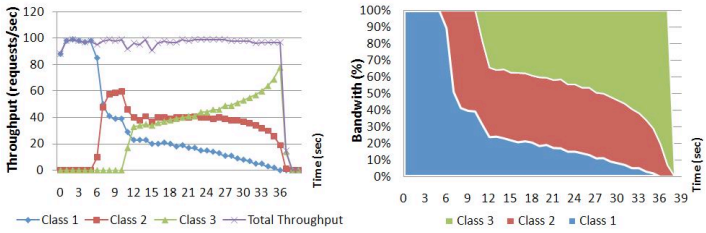

Figure 4. CBFQ results

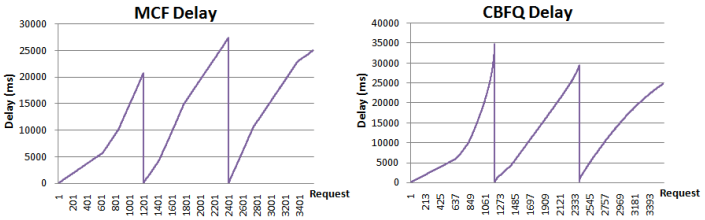

Figure 5. Service Delays

Figure 3 and Figure 4 present the test monitoring results for MCF and CBFQ algorithms respectively, regarding the achieved throughput and bandwidth occupancy. Figure 5 presents the evolution of the metric delay in both cases.

MCF results presented in Figure 3 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 by both active queues proportionally to its 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 4 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, queues 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 5 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 hold 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.

V. Conclusions

This paper has presented an admission control solution that implements the recently proposed scheduling disciplines Most Credit First (MCF) and Credit Based Fair Queuing (CBFQ).

The deployment of this solution in a service provisioning platform has allowed to regulate the throughput of distinct service classes, according to each class priority, and to support the differentiation of promptness level provided among the services requests. This solution is a step towards quality of service provisioning through differentiated admission control, which is essential to enhance the performance and reliability of the platform. The preliminary results have shown that the behavior of CBFQ is based on performing a fair scheduling while maintaining queues lengths balanced. MCF performs a fair scheduling exclusively based on queues priorities, without considering the level of congestion present in the queues.

Future work intents to further explore the behavior of both algorithms under distinct test scenarios and enhance the proposed admission control solution.

References

[1] Briscoe, B., "*Flow rate fairness: dismantling a religion.*", SIGCOMM Comput. Commun. Rev., 2007. 37(2): p. 63-74.

[2] Rexford, J.L., A.G. Greenberg, and F.G. Bonomi, "*Hardware-Efficient Fair Queueing Architectures for High-Speed Networks*", in *In Proceedings of INFOCOM*. 1996. p. 638-646.

[3] Wong, W.K., H.Y. Tang, and V.C.M. Leung, "*Token bank fair queuing: a new scheduling algorithm for wireless multimedia services: Research Articles*". Int. J. Commun. Syst., 2004. 17(6): p. 591-614.

[4] Pedro Sousa, Paulo Carvalho, and V. Freitas, "*A multi-constrained QoS aware scheduler for class-based IP networks*". 4TH Symposium on communications Systems, Networks and Digital Signal Processing, 2004.

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

[6] Eryilmaz, A. and R. Srikant, "*Fair resource allocation in wireless networks using queue-length-based scheduling and congestion control*". IEEE/ACM Trans. Netw., 2007. 15(6): p. 1333-1344.

[7] Jalali, A., R. Padovani, and R. Pankaj, "*Data throughput of CDMA-HDR a high efficiency-high data rate personal communication wireless system*". Vehicular Technology Conference Proceedings, 2000. 3: p. 1854--1858.

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

[9] Trajkovic, N.A.a.B.C.a.L., "*Modeling Packet Scheduling Algorithms in IP Routers*". 2001: School of Engineering Science, Simon Fraser University.

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

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

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

[13] Demers, A., S. Keshav, and S. Shenker, "*Analysis and simulation of a fair queueing algorithm*". SIGCOMM Comput. Commun. Rev., 1989.

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

[15] Golestani, S.J., "*A self-clocked fair queueing scheme for broadband applications*". NFOCOM apos;94. , 1994. 2.

[16] Pan D., Y.Y., "*Credit based fair scheduling for packet switched networks*". INFOCOM 2005, 2005. 2.

[17] 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. Netw., 2001. 9(5): p. 591-604.