Dynamic Service Contract Enforcement in Service-Oriented Networks

Yesid Jarma, Keerthana Boloor, Marcelo Dias de Amorim, Yannis Viniotis, and Robert D. Callaway

Abstract—In recent years, Service Oriented Architectures (SOA) have emerged as the main solution for the integration of legacy systems with new technologies in the enterprise world. A service is usually governed by a client service contract (CSC) that specifies, among other requirements, the rate at which a service should be accessed, and limits it to no more than a number of service requests during an observation period. Several approaches, using both static and dynamic, credit-based strategies, have been developed in order to enforce the rate specified in the CSC. Existing approaches have problems related to starvation, approximations used in calculations, and rapid credit consumption under certain conditions. In this paper, we propose and validate DoWSS, a doubly-weighted algorithm for service traffic shaping. We show via simulation that DoWSS possesses several advantages: it eliminates the approximation issues, prevents starvation and contains the rapid credit consumption issue in existing credit-based approaches.

Index Terms—Service-oriented networks, Web services, service traffic shaping, contract enforcement, middleware appliances, appliance cluster, credit-based algorithm

1 Introduction

During the last few years, the rise of the Internet has changed the way business is conducted worldwide. To remain competitive, businesses have been implementing information technology support for business processes over the years. However, budgetary issues, the continuous growth of the organizations, the heterogeneity of existing systems, among others, increase the complexity of deployment and integration of new technologies. In this context, Service-Oriented Architectures (SOA) have become the main solution for the integration of applications and technologies in the business domain and for the collaboration among industrial partners. A way to implement SOA is through the concept of Web Services (WS) [1]. These are software systems designed to support machine-to-machine interoperability through a set of Extensible Markup Language (XML) based open standards, such as Web Services Description Language (WSDL) [2], Simple Object Access Protocol (SOAP) [3], and Universal Description, Discovery and Integration (UDDI) [4].

Even though the use of XML-based standards allows easy integration with external data sources, one of the major issues preventing wider adoption of Web Services is performance [5]. Indeed, as the time needed to parse an XML document can take up to a few minutes [6], the response time of a Web Service is potentially large. To better satisfy business goals, service providers use specific hardware that provides accelerated XML processing called Service-Oriented Networking (SON) appliances [7].[1] Besides processing XML documents, enabling security and integrating with legacy systems, SON appliances may also be responsible for controlling the rate at which documents are sent to the service hosts (i.e., they shape the traffic). A service is typically governed by a Client Service Contract (CSC) dictated by a Service Level Agreement (SLA). The CSC specifies, among others, a Service Access Requirement (SAR), which is the rate at which the services may be accessed in order to prevent them from being overwhelmed. The SAR is usually defined as: "Limit the rate to a service provider to no more than X requests per second with an observation/enforcement period of T seconds". In other words, the maximum number of requests sent to the service provider within the period T is $C = X \times T$ (see details in Section 2).

Traffic shaping is a well-known classic problem in network traffic engineering [8], [9], [10]. Nevertheless, in Service-Oriented Networking the problem is fundamentally different. The main difference is that, unlike classic networking where traffic shaping is done point-to-point, in SON service clients usually access services from multiple access points. Furthermore, in SON, multiple SON appliances can be used to address issues such as security, fault tolerance, and performance. Therefore, the key challenge is how to enforce the CSC by taking local actions at each appliance.

- *Y. Jarma and M. Dias de Amorim are with LIP6/CNRS - UPMC Sorbonne Universités, 4 Place Jussieu, 75005 Paris, France. E-mail: {jarma,amorim}@npa.lip6.fr*
- *Y. Vinniotis and K. Boloor are with the Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC 27695. E-mail: candice,kboloor@ncsu.edu*
- *R. Callaway is with IBM, 4205 S Miami Blvd, Durham, NC 27703. E-mail: rcallawa@us.ibm.com*

Manuscript submitted on February 2011.

1. SOA is a software architecture for building applications that implement business processes or services by using a set of loosely coupled, black-box components orchestrated to deliver a well-defined level of service. SON is an emerging architecture that enables network devices to operate at the application layer with features such as offloading, protocol integration, and content based routing.

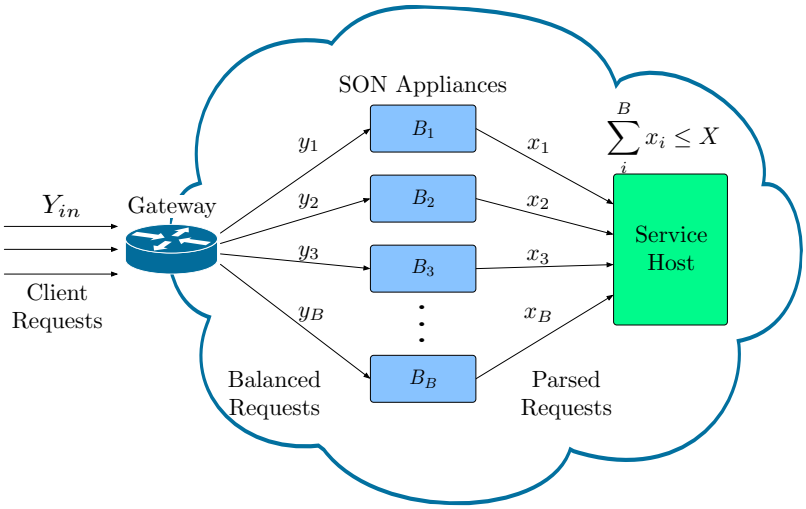


Fig. 1. System Architecture showing the different elements and parameters involved in the service contract enforcement procedure.

The typical solution consists of using a manual and static allocation strategy, in which the allowed rate is equally divided among all the access points. This solution, although simple, is quite inefficient as it only provides satisfactory performance when the incoming traffic rates at the SON appliances are identical. In a companion work [11], we proposed a better, more dynamic solution that monitors the traffic on a regular basis and adapts the rate by reassigning credits to each appliance under a weighted strategy based on queue sizes.

In order to comply with the CSC, the calculations used in existing credit-based approaches lead to three main drawbacks:

1) *Flooring effect.* Existing credit-based solutions require the use of a flooring function to approximate the results to the integer immediately below. In some cases, when the number of appliances is not a divisor of the available credits, the use of a flooring function leads to under-utilization of the system.
2) *Fast start.* When the system operates under high input rates, all the available credits are rapidly consumed early in the enforcement period. This may result in overwhelming the service host, since a large number of requests are being sent during a time period substantially smaller than the specified enforcement period.
3) *Starvation.* The weighted strategies used for dynamic credit allocation are based on queue sizes. As a consequence, the appliances with at least one queued event may be allocated all the credits, thus depriving the appliances with empty queues from credits[2].

2. Note that the definition of starvation used throughout this paper differs from that used in scheduling literature, where a process is perpetually denied necessary resources, and therefore can never finish its task [12].

The immediate repercussion of these issues is that they lead to suboptimal performance. *Given the costs of implementing SON and issues inherent to the provision of Web Services, it is imperative to design efficient algorithms that optimize the overall utilization of the system.*

In order to solve the issues cited above, we propose DoWSS (**Do**ubly-**W**eighted algorithm for **S**ervice traffic **S**haping), an algorithm for service traffic shaping. Our approach is based on dividing the enforcement period into several smaller *enforcement subperiods* and the calculation of maximum allowed rates avoiding the use of flooring functions. By using a doubly-weighted strategy, our approach prevents starvation issues that may appear under certain conditions. We also introduce a procedure to contain the rapid consumption of credits at the beginning of the enforcement period, when there are high input rates to the system. Through simulation analysis, we show that our approach not only outperforms existing approaches, but it also has a substantial positive impact on the overall performance of the system over time.

In summary, the contributions of our work are:

- We identify three issues present in existing credit-based service traffic shaping approaches: *flooring*, *starvation* and *fast start*.
- We propose a dynamic, doubly-weighted, credit-based approach, that avoids the *flooring* and *starvation* issues, and uses system resources to their fullest.
- We introduce a contention mechanism to minimize the *fast start* phenomenon, which manifests itself when the input rate to the system is much greater than the Service Access Requirement rate.

The remainder of this paper is structured as follows. In Section 2, we introduce the system architecture we focus on. In Section 3, we describe the context and system architecture, and perform a detailed analysis

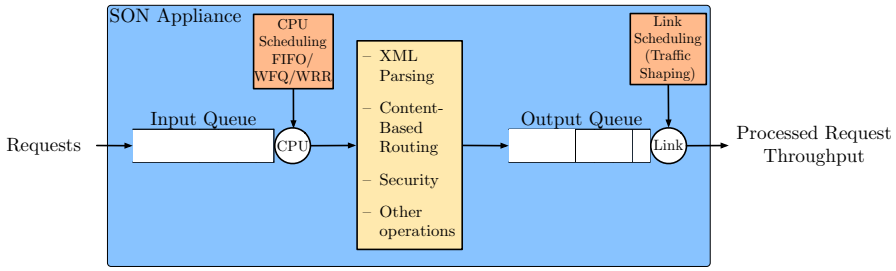


Fig. 2. Internal Architecture of a SON Appliance showing some of its basic elements and functions.

of existing issues in service traffic shaping in SON. In Section 4, we introduce our approach, while in Section 5 we evaluate our algorithm via extensive simulations. We discuss our results and point out some open issues in Section 6. Related work is presented in Section 7. Finally, we conclude the paper and give insights about future work in Section 8.

2 System Architecture

Service-Oriented Architectures are typically centralized systems in which one node executes and manages instances of one or more services. However, to address possible scalability issues, the centralized service may be replicated and the requests balanced among the replicas [13]. The general architecture of the considered system is depicted in Fig. 1. There are four main elements composing this architecture:

- **Clients:** The clients are nodes that generate service requests. They can be located anywhere in the Internet. In Fig. 1, Y_{in} denotes the input rate of requests into the system.
- **Gateways:** These are border routers. They are responsible of forwarding service requests to the SON Appliances, and distributing the service load among the appliances without any deep-content inspection.
- **SON Appliances:** These are middleware responsible for translating XML requests into the system's local language. They are also responsible for controlling the rate at which the service requests are forwarded to the service host. In the figure, x_i denotes the number of processed requests appliance B_i sends to the service host within some time interval (to be defined in Section 3). As it will become clearer later on in this paper, we use "numbers of requests" instead of "rate" to simplify the integration of the service contract into the algorithm.
- **Service host:** This node handles processing of the requests. It also specifies the rate at which the services may be accessed or Service Access Requirement (SAR). In the figure, the service may not receive more than C requests during an time interval of duration T, where $C = X \times T$.

A service is typically accessed from a single SON Appliance; therefore, the traffic from the gateway to the service host follows a point-to-point pattern. A single entry point provides the advantage of simplified service access management. Furthermore, since point-to-point traffic shaping is a well-studied problem in the networking space, well-known solutions from packet/ATM networks can be applied.

Nevertheless, in the SON environment, clients may access services from multiple entry points. The existence of multiple entry points may be dictated by security policies (the presence of multiple security zones), robustness (fault tolerance), or performance requirements (load is balanced on a cluster of SON appliances). SON appliances can implement a number of functions, which include functional offloading, service integration, and intelligent routing [14]. In addition to providing these functions, SON appliances are also responsible for controlling the rate at which client requests are sent to the service hosts. This problem is known as the *service traffic shaping* problem.

2.1 Scheduling vs. Shaping

Fig. 2 shows the internal architecture we consider for a SON appliance. Recall from the previous section that, besides SON-related processing of XML documents, an appliance is also responsible for enforcing the SAR requirement, i.e., for limiting the rate at which processed documents are sent to the service hosts to any desired rate X, *regardless of the input rate*. In our assumed model, requests entering each appliance are placed in an input queue. A CPU performs all the SON-related tasks. A CPU scheduler determines the order in which requests from the input queue get allocated to CPU resources, by using a (work-conserving) job scheduling algorithm such as FIFO, WFQ [15] and WRR. Once the requests have been processed, they are placed in an output queue, to be transmitted to the service tier via a communication link. A Link scheduler determines the order in which processed requests access the outgoing link. In our problem, we have only one class of requests, so FIFO ordering will suffice. *The SAR requirement is enforced by the Link scheduler*; work-conserving algorithms are not suitable for such enforcement, since they do not limit the output rate. Non-work-conserving algorithms must be used for the control of the outgoing link. Suppose, for example, that the SAR specified $X = 2$, the input rate was 4 and the outgoing link had a capacity higher than

the input rate. A work-conserving scheduling algorithm (e.g., WFQ) would not be able to enforce this SAR. For clarity and in accordance with jargon from networking environments, we label this function in Fig. 2 as Traffic Shaping.

In this paper, we specifically consider the service *traffic shaping* problem where several access points (i.e., SON Appliances) access concurrently a single service host (i.e., multipoint-to-point case). We formalize the problem in more detail in the following section.

3 Problem Definition

3.1 CSC Enforcement

Services are governed by a Client Service Contract (CSC) dictated by a Service Level Agreement (SLA). The CSC specifies, among others, a Service Access Requirement (SAR), which is the rate at which the services may be accessed in order to prevent them from being overwhelmed. It is worth noting that, there are other types of requirements in typical SLAs, like delay-related requirements, which are addressed via work-conserving algorithms such as Weighted Fair Queuing [15]. The SAR SLA is a special case of a traffic shaping problem that necessitates the use of non-work-conserving algorithms.[3] In our architecture, in addition to providing the functions previously described, SON appliances are responsible for enforcing the CSC.

Traffic shaping is a well-known classic problem in network traffic engineering [8], [9], [10]. However, in Service-Oriented Networking the problem is fundamentally different. In classic networks, the resource protected by the shaping function is typically link bandwidth and buffer space, the units of which are precisely defined and measurable. Service Level Agreements are standardized by industrial bodies and CSC contracts are very well defined. In SON, the resource protected by the shaping function is CPU processing power. Moreover, *CSC contracts are not precisely defined and measurable.*

We are interested, in particular, in the SAR definition, which in general follows the following format: "Limit the rate to a service provider to no more than X requests per second with an observation/enforcement period of T seconds", where an enforcement period is a time interval during which the aggregate of requests sent to the service host by all the appliances cannot exceed $C = X \times T$. In this particular case, since "requests" are defined in units of XML documents, CPU processing time is not known exactly. Furthermore, this SAR does not include additional requirements such as a maximum burst size. On the other hand, in traditional networks, the parameters for implementing token buckets, for example, include, in addition to an average rate, a peak rate (which is the maximum rate at which packets can be sent in a short time interval), and a burst size (a limit for the number of packets to be transmitted in a short time interval). Finding a way to establish other types of CSCs that are well-defined remains an open issue (cf. Section 6.3).

3. For example, consider the case of a single appliance with very high processing capacity, a single client that sends traffic at rate 2, and a desired limit equal to 1. A work-conserving scheduling algorithm would not be able to limit the rate.

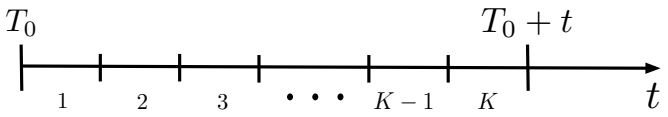


Fig. 3. Division of the enforcement period T into K smaller subperiods.

Another fundamental difference is that in the classic networking environment, traffic shaping has local scope, since traffic is in the form of a single connection. In the SON environment, service clients access services from multiple entry points. The existence of multiple entry points may be dictated by policy (e.g., the presence of multiple security zones) or performance requirements (e.g., clusters of SON appliances); the desired effect is "global" shaping. *The challenge is therefore to enforce the traffic contract by taking local actions at each entry point.*

3.2 Enforcement Strategies

We have so far identified in the literature two different strategies for enforcing the CSC in SON. The simplest strategy is to use a manual and static allocation (MSA), in which the allowed rate is equally divided among all the SON appliances:

$$x_i = \left\lfloor \frac{X \times T}{B} \right\rfloor, \quad \forall i \in [1, \ldots, B], \tag{1}$$

where x_i is the number of credits allocated to appliance i, X is the maximum rate allowed per enforcement period, T is the duration of the enforcement period, and B is the number of appliances. This solution, although simple, is quite inefficient as it only provides satisfactory performance when the incoming traffic rates at the appliances are identical. Therefore, a number of appliances may hold queued requests while others remain idle.

In a previous work, we introduced CASTS, a solution that relies on the communication and processing capabilities of the appliances in order to provide a better response to the requirements specified in the CSC [11]. In summary, CASTS works as follows. We proposed dividing the CSC enforcing period into K *subperiods* (see Fig. 3), during which the traffic is measured and the rate adapted also by means of assigning credits to the appliances. During subperiod k, each appliance estimates the number of credits it will have during the next interval using queue sizes, measures the number of requests queued and already sent, and broadcasts these values to the other appliances. Each appliance updates its shaping rate for subperiod $k+1$ after receiving the information from all the other appliances as follows:

$$x_i(k+1) = \left\lfloor D \times \frac{Q_i(k)}{\sum_{j=1}^{B} Q_j(k)} \right\rfloor, \quad (2)$$

where $Q_i(k)$ is the number of queued requests at appliance i, D is the number of remaining credits of the enforcement period, and B is the number of appliances. Note that an approximation function (in this case, a *flooring* function) is necessary, as the CSC specifies an *integer* number of documents to be sent to the service tier. This solution guarantees that the SAR is respected at all times by assigning credits dynamically under a weighted strategy. This approach uses MSA to adjust the number of credits during the first subperiod and when there are no queued requests in any appliance.

3.3 Issues

In this paper, we identify and address three problems associated with existing service traffic shaping solutions. We start by defining them in the following.

3.3.1 Flooring Effect

CSCs specify the SAR in terms of requests per time unit within the observation period. Consequently, in order to comply with the CSC, the calculations conducted in both solutions involve the use of a *flooring function* to approximate the number of allocated credits to the integer immediately below. In some cases, when the number of appliances is not a divisor of the number of available credits, the use of a flooring function leads to under-utilization of the system. Fig. 4 depicts a sample of the typical performance of CASTS and MSA for different input rates (Y_{in}), with an enforcement period (T) of 1 second, using ten SON Appliances and a maximum allowed rate (X) of 128 requests per second represented by a horizontal dotted line. Even though both approaches process a number of requests near $C = X \times T = 128$, they never reach the maximum value. Therefore, at each enforcement period, there are a number of requests that are left unprocessed and accumulate significantly over time (see Fig. 8). As a consequence, the system is unable to exploit its maximum capacity. *Given the costs of implementing SON and issues inherent to the provision of Web Services, it is imperative to design efficient algorithms that optimize the overall utilization of the system.* Achieving optimal performance is fundamental in the long term.

Definition 1. Let $R(K)$ be the total number of processed requests within the observation period T, $X \times T$ be the maximum allowed number of requests to be sent to a service host during an observation period T, and $Y_{in} \times T$ be the total number of requests generated within an observation period T. We say that a shaping algorithm is *optimal* if $R(K) = \min[X \times T; Y_{in} \times T]$.

3.3.2 Starvation

Even though the gateway performs some sort of load balancing, this does not guarantee that the load will be equally distributed among all appliances. Indeed, as requests require different processing times, load balancing at the gateway is not transferred to the output rates of the SON appliances. As a consequence, since both MSA and CASTS are based on the allocation of credits that the appliances will use to send requests to the service hosts, a starvation phenomenon appears, in which some appliances will have available credits, while others remain idle. This phenomenon is evidenced in two different ways, depending on the credit allocation scheme used.

When using MSA, an appliance is allocated a number of fixed credits, and it can use them to send requests to the service host as fast as it can. Furthermore, since this method does not involve any kind of communication between appliances, an appliance is not able to know if the other appliances have queued requests. Consequently, when it operates under high (instantaneous) input rates, an appliance receiving more requests will rapidly consume all of its credits, while other appliances receiving a smaller amount of requests will have more credits available even though they will not use them.

CASTS relies on communication between appliances, and each appliance informs the others of the number of credits it has consumed so far and the number of request that it has still queued. The appliances use this information to perform a weighted assignment of credits, in which the appliances with the largest queues will get the most credits. Nevertheless, this scheme penalizes the appliances with little or no queued requests. The weighted allocated scheme used in CASTS attempts to estimate the sending rate to the service host during the following interval using queue sizes. However, it cannot predict how many requests will enter the system during the next interval, and if these new requests will be equally balanced among all appliances. Therefore, it is possible that an appliance with a larger queue and more credits will continue to accumulate requests without being able to process them; at the same time, an appliance with a small queue that was allocated a few credits will consume these few credits before receiving new requests, remaining idle during the entire subperiod even though it is capable of processing additional requests.

3.3.3 Fast Start

An observed phenomenon in both approaches is the high sending rate of requests to the service host during a small period of time at the beginning of each enforcement period. We refer to this phenomenon as *fast start*. MSA and CASTS use different methods for allocating credits to the appliances. Nevertheless, the allocation does not specify *how fast* the credits should be used. Therefore, when there are high input rates, the appliances will rapidly consume all their credits during the first moments of the enforcement period by processing and sending requests to the service hosts as fast as possible. Even though the idea behind the CSC is to prevent the service hosts from being overwhelmed,

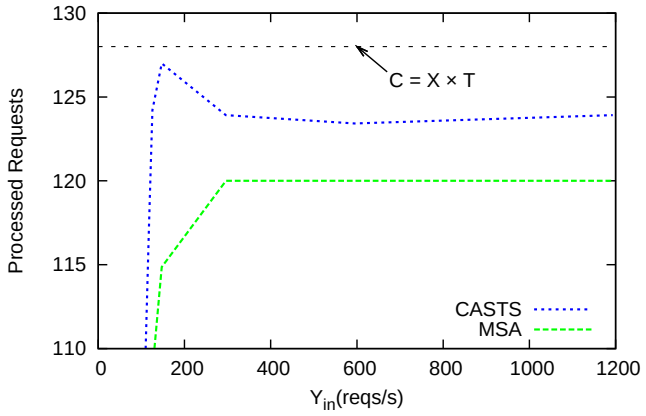


Fig. 4. Performance of CASTS and MSA for different input rates and $X = 128$ reqs/sec.

sending a large number of requests during a time period substantially smaller than the specified enforcement period could result in effectively overwhelming the service host.

This type of behavior is also associated with the ill-defined nature of the CSCs in SON. In the ATM network environment, for example, the CSCs include the SAR definitions for the Maximum Burst Size (MBS) and a Cell Delay Variation Tolerance (CDVT). In this way, the number of cells per time period arriving at a host is easily limited. Since in a SON environment the CSCs do not incorporate equivalent definitions of burst sizes and maximum delays, undesirable effects such as the *fast start* appear, resulting in a substantial impact on the overall efficiency of the system.

It is worth noting that, even though *fast start* is an undesired phenomenon, it is nonetheless inherent to the way these algorithms work. Indeed, since both allocation methods are based on distributing all available credits among all SON appliances, the *fast start* will depend directly on the maximum allowed rate to enforce and the number of SON appliances used. For instance, suppose that we have $B = 32$ SON appliances and that the SAR rate specified is $X = 128$ reqs/sec. When the instantaneous input rate is much larger than the SAR rate ($Y_{in} \gg X$, e.g., $Y_{in} = 3000$ reqs/sec), even if each SON appliance is limited to forward only one request per enforcement subperiod, the entire system will nevertheless send 32 requests to the service host during the first enforcement subperiod. Therefore, the service host will receive 25% of the maximum allowed rate during the first subperiod alone.

3.4 Discussion

The immediate consequence of these issues is that they lead to "suboptimal performance". The main issue present in existing credit-based approaches is the use of flooring functions to approximate the number of allocated credits to the integer immediately below. In order to avoid this issue, another kind of approximation function should be used. The *starvation* issue is mainly due to the use of a weighted strategy based only on the number of queued requests. This issue could be avoided by taking a more thorough look on the characteristics of the queued requests. Since the *fast start* issue is inherent to the way credit-based algorithms work, it is not possible to completely avoid it. Nevertheless, a mechanism that partially contains the effects induced by the *fast start* could be designed and incorporated into the overall management system.

4 DoWSS (Doubly-Weighted Algorithm for Service Traffic Shaping)

To address the existing issues in CSC enforcement in Service-Oriented Networks, we propose DoWSS, a doubly-weighted algorithm for service traffic shaping in service-oriented networks. Our approach is especially designed for Web Server farms implementing the multipoint-to-point access strategy. In this section, we first specify the preliminaries of our approach. The details of DoWSS come thereafter.

4.1 Preliminaries

DoWSS is based on the notion of *enforcement subperiod* introduced in CASTS [11]. The enforcement period is divided into K subperiods (see Fig. 3). During each subperiod, the algorithm will measure the number of requests that were processed and forwarded to the service host and queue sizes, and adapt its sending rate for the next subperiod by assigning *credits* to each appliance. A *credit* allows an appliance to send a processed request to the service host.

CSCs specify the SAR in requests per second within the enforcement period. Consequently, credit-based approaches must approximate the number of allocated credits by an integer value. The main difference between DoWSS and existing credit-based approaches is the type of function used to approximate the number of allocated credits to the integer. Instead of using a flooring function, like other approaches, DoWSS uses a ceiling function. Even though the use of a ceiling function may lead to a non-compliance of the maximum request rate, the CSC can still be enforced by using the communication capacities of the appliances. Since the SAR is a fixed value, all SON Appliances have this information beforehand. Therefore, when calculating the rate allocated to each appliance at each subperiod, the non-compliance of the CSC can be detected. If the credits that are to be allocated to each appliance exceed the number of remaining credits, one or several appliances will be *penalized* by having credits reduced, so that the CSC is respected. This penalization is done randomly, and the procedure varies depending on whether or not there are queued requests in any appliance.

4.2 Case I: Empty Queues

First, the number of credits/requests available for allocation is calculated:

$$D = X \times T - R(k-1), \tag{3}$$

where $R(k-1)$ is the number of requests processed up until the latest subinterval. By definition, $R(0) = 0$. At the beginning of the first enforcement subperiod, and in subperiods where there are no queued requests in any appliance, MSA is used for credit allocation:

$$x_i(1) = \left\lceil \frac{D}{B} \right\rceil, \quad i = 1, \ldots, B. \tag{4}$$

Once each appliance has calculated the number of credits allocated to it, we compute P, which is the number of credits exceeding the CSC:

$$P = (x_i(k) \times B) - D. \tag{5}$$

If $P > 0$, appliance B_i will generate then a random number N_i. This number is broadcast to the rest of the appliances. When all the appliances finish this information exchange, each appliance i will find the P lowest numbers among the numbers generated by all the appliances. If its own randomly generated number N_i is among the P lowest, the appliance has one of its credits removed. Otherwise, the appliance keeps all of its assigned credits. Note that this exchange of random values can be done both in a centralized or distributed manner. By design choice, in this paper we opt for the distributed way.

To reduce the possibility of conflicts between appliances (i.e., two or more appliances generating the same number), N should be chosen in a range much larger than the number of appliances. Nevertheless, it is still possible for two appliances to generate the same random number (see details in Section 4.4).

4.3 Case II: Non-empty Queues

When there is at least one queued request in the entire system, the use of the weighted strategy proposed in [11] may lead to *starvation* in appliances with empty queues (cf., Section 3.3.2). To avoid this, a doubly-weighted strategy is proposed. First, the n-th request in the queue is assigned a weight w_n, calculated as

$$w_n = \log_{10}\left(\frac{T \times V}{K \times s_n}\right), n = 1, \cdots, Q, s_n \neq 0, \tag{6}$$

where V is the processing speed of the appliance in bits per second and s_n the size of the n-th request, measured in bits. For simplicity, we make the assumption that, on average, the processing time of a request is proportional to the length (size) of the request[4]. The weight of a request is therefore inversely proportional to its size and depends directly on the number of measurement subperiods used. Therefore, large requests, which take longer to process, will have smaller weights. When the request processing time is larger than T/K, the weight of the request is negative. In this case, the weight of the request is set to zero. If an appliance has an empty queue, the weight is calculated using a virtual file size of 1 bit. We use a logarithmic scale in our calculations in order to work with numbers in a smaller range. The weight of appliance B_i is the sum of the weights of all the requests in the queue:

$$W_{B_i} = \sum_{n=1}^{Q_i} w_n. \tag{7}$$

Once each appliance calculates its own weight, it calculates the number of credits it is allocated during the next subperiod under a weighted strategy:

$$x_i(k) = \left\lceil D \times \frac{W_{B_i}(k)}{\sum_{n=1}^{B} W_n(k)} \right\rceil. \tag{8}$$

Since the number of allocated credits is calculated locally, each appliance broadcasts this value to the other appliances. Upon reception of the information coming from other appliances, each appliance calculates the number of exceeding credits P:

$$P = \left(\sum_{n=1}^{B} x_n(k)\right) - D. \tag{9}$$

Then, in order to specify which appliances are to be penalized, the same procedure used with empty queues is used. Each appliance will generate a random number N. This number is broadcast to the other appliances. Once an appliance receives all the information coming

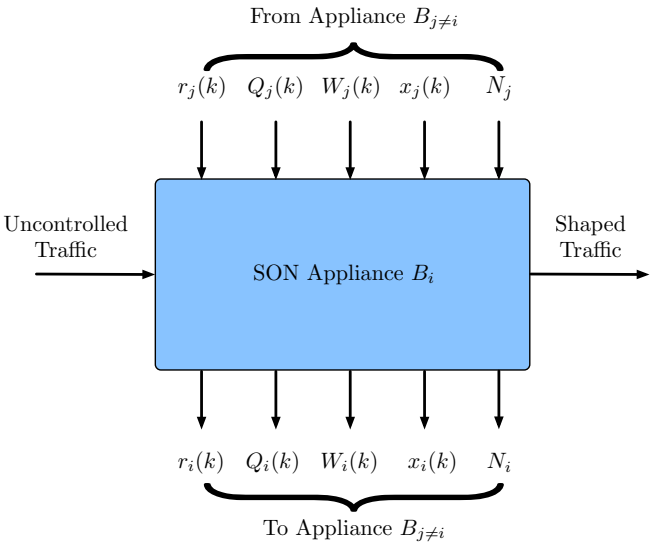


Fig. 5. DoWSS information exchange during the k-th enforcement subperiod.

4. In reality, the average processing time is proportional to length of the requests (e.g., due to parsing the entire XML document for checking well-formedness) as well as other factors, like the actual content of the XML document.

Algorithm 1 DoWSS algorithm.

Input: $k, N_j, \forall j \in [1, \ldots, B], j \neq i$.
Input: (When $1 < k \leq K$) $r_j(k-1)$, $Q_j(k-1)$, $W_j(k-1)$, and $x_n(k)$, $\forall j \in [1, \ldots, B], j \neq i$.
Output: new count $x_i(k)$.

if $k = 1$ **then**
  $x_i \leftarrow \left\lceil \frac{X \times T}{B} \right\rceil$
  $P = (x_i \times B) - (X \times T)$
  $N = \{N_1, \ldots, N_B\}$
  if $P \neq 0$ **then**
    Find P lowest of the set N
    if N_i is among P lowest **then**
      $x_i(k) \leftarrow x_i(k) - 1$
    end if
  else
    $x_i(k) \leftarrow x_i(k)$
  end if
else
  $r(k-1) = \sum_{j=1}^{B} r_j(k-1)$
  $R(k-1) = \sum_{n=1}^{k-1} r(n)$
  $W(k-1) = \{W_1, \ldots, W_B\}$
  $N = \{N_1, \ldots, N_B\}$
  if $R^j(k) < X^j$ **then**
    $D \leftarrow X \times T - R(k-1)$
    if $\sum_{n=1}^{B} Q_n(k) = 0$ **then**
      $x_i(k) \leftarrow \left\lceil \frac{D}{B} \right\rceil$
      $P = (x_i(k) \times B) - D$
      if $P \neq 0$ **then**
        Find P lowest of the set N
        if N_i is among lowest **then**
          $x_i(k) \leftarrow x_i(k) - 1$
        end if
      else
        $x_i(k) \leftarrow x_i(k)$
      end if
    else
      $x_i(k) \leftarrow \left\lceil D \times \frac{W_i(k)}{\sum_{n=1}^{B} W_n(k)} \right\rceil$
      $P = \left(\sum_{n=1}^{B} x_n\right) - D$
      if $P \neq 0$ **then**
        Find P lowest of the set N
        if N_i is among P lowest **then**
          $x_i(k) \leftarrow x_i(k) - 1$
        end if
      else
        $x_i(k) \leftarrow x_i(k)$
      end if
    end if
  else
    $x_i(k) \leftarrow 0$
  end if
end if

from other appliances, it is penalized if its own randomly generated number N_i is among the lowest P. Fig. 5 depicts the information exchange during enforcement subperiod k. The entire procedure is summarized in Algorithm 1.

4.4 Conflict Resolution

To reduce the possibility of conflicts between appliances at credit removal, as explained before, random numbers are selected in a range that is much larger than the number of appliances. Nevertheless, it is still possible that two or more appliances generate the same random number. To avoid conflicts in this situation, the following procedure is followed. If all the "conflicting" numbers generated are among the P lowest random generated numbers, all involved appliances have their exceeding credits removed.

However, the conflicting generated numbers might be the greatest of the P lowest random generated numbers. For example, suppose that $B = 3$ is the number of appliances, $P = 1$ is the number of exceeding credits and $N = [100, 264, 264]$ is the set of random numbers generated by appliances 0, 1 and 2 respectively. In this particular case, the number of randomly generated numbers to penalize will exceed P, since appliances 1 and 2 generated the same number and it is greater than the number generated by appliance 0.

When this kind of situation occurs, a concurrency mechanism is used. When an appliance detects a value received from the other appliances equal to the value it has generated, it will notify the conflicting appliances of the equality and tell them to keep its assigned credit. To avoid the possibility of two appliances transmitting this notification at the same time and therefore causing further conflicts, this transmission is delayed by using a random timer. The first appliance whose timer expires, notifies the conflicting appliances, and will thus be effectively penalized.

4.5 Addressing the *fast start* issue

We introduce in our approach an optional method for avoiding the *fast start* issue (cf., Section 3.3.3). The idea is to further limit the number of requests sent to the service host during the beginning of each enforcement period, and distribute them over the entire observation period. Nevertheless, since this phenomenon only appears when the instantaneous entry rate exceeds X by a great value, the system will only activate the contention mechanism under these conditions.

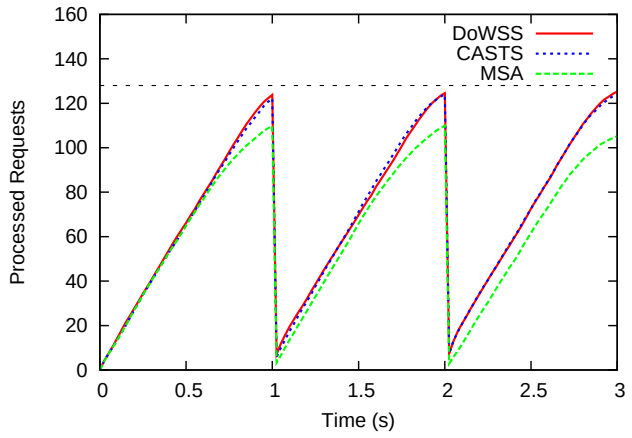
At the beginning of each subperiod, the system can estimate the global input rate Y_{in} by measuring the number of received requests during the last subperiod. When a high input rate is detected, each appliance can calculate F, the "limited" sending rate towards the service hosts:

$$F = \left\lceil \frac{T \times X}{K \times B} \right\rceil. \tag{10}$$

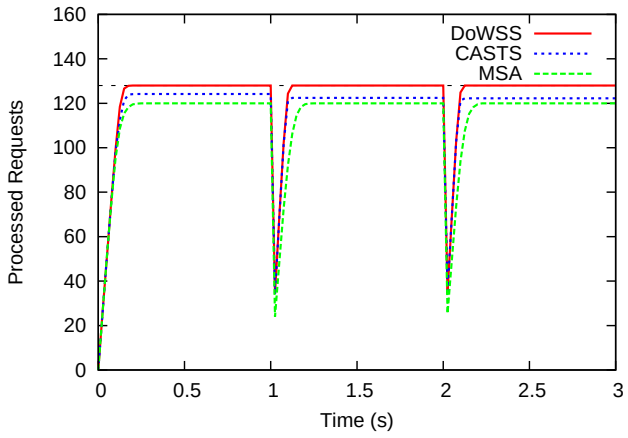
Each appliance will then adjust its number of allocated credits to F for the subperiod, therefore $x_i(k) \leftarrow F$. Note that, with this optional method, by performing another manual rate assignment, the desired weighted effect is almost lost. If during the enforcement period the traffic abruptly changes from subperiod to subperiod (e.g., in the presence of bursty traffic), the contention method is turned off during the periods where the input rate is not greater than the SAR.

5 Evaluation

To study the performance of DoWSS, we undertook a series of simulations. To this end, the OMNeT++ Discrete Event Simulation System [16] was used. The OMNeT++ library controls the simulated time and the concurrent execution of the code running on each one of the simulated SON Appliances. All appliances run the same code. The algorithms are written in C++ and are event driven.



(a) $Y_{in} = 128$ reqs/s.



(b) $Y_{in} = 1000$ reqs/s.

Fig. 6. Number of processed requests per enforcement period (T) for $Y_{in} = X$ and $Y_{in} >> X$ under uniform traffic

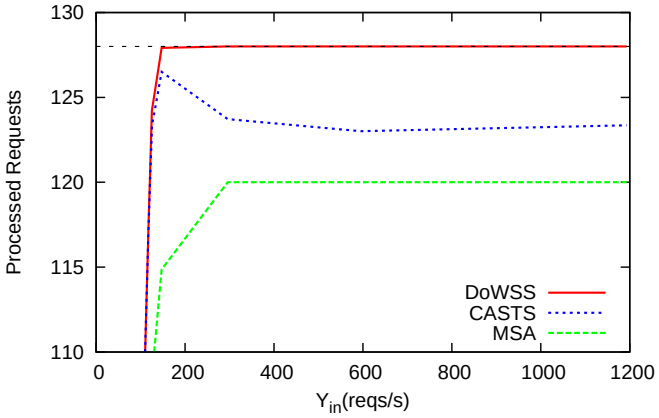


Fig. 7. Total number of processed requests over one enforcement period for different input rates.

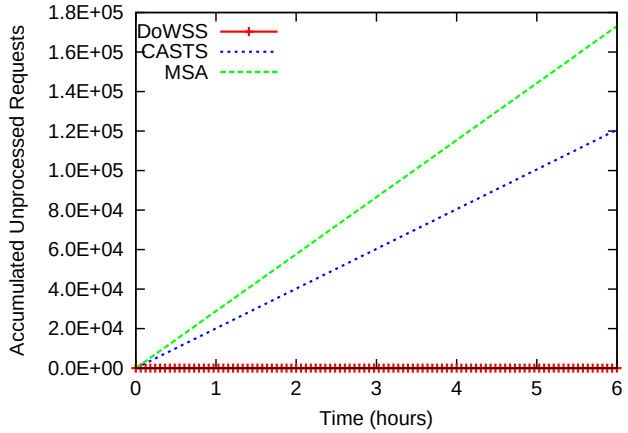


Fig. 8. Total number of unprocessed requests over a period 6 hours for $Y_{in} = 300$ reqs/sec.

5.1 Experimental Setup

The simulation-based evaluation of DoWSS is centered around answering three questions:

1) Is the performance of DoWSS *optimal*?
2) Are the techniques used in DoWSS able to solve the *Flooring, Fast Start and Starvation issues?*
3) If input rates vary, will DoWSS be able to *adapt* to these variations while continuing to provide an *optimal performance*?

The first set of simulations aims to answer questions 1) and 2). For this set, the client service requests are modeled as Poisson processes. The average input rate to the system, noted as Y_{in}, is chosen as a fixed value (unknown, of course, to the SON Appliances) and is varied to verify CSC compliance for all input rates. The second set of simulations answers the 3rd question. To this end, we simulate bursty traffic using a Poisson Pareto Burst Process (PPBP) model [17]. Bursts are modeled as Poisson processes with a duration sampled from a Pareto distribution. In both sets, the processing rate of each document at each appliance varies and depends directly on document sizes. In a previous work [11], we explored the responsiveness of credit-based algorithms. We observed that for $T = 1$ and $K = 40$, the algorithm achieves a reasonable responsive behavior[5]. Therefore, for all the presented simulations, we have set $T = 1$ second, $K = 40$ and $X = 128$ requests per second, unless otherwise specified. All data points shown on the curves represent an average over 100 runs. We have calculated confidence intervals for each data point, however we do not show them on the curves for simplicity.

5.2 Performance Metrics

In order to properly measure the performance of DoWSS and compare it with other existing approaches, we use the following performance metrics:

5. In a real deployment scenario, the choice of the length of an enforcement period rests at the discretion of an IT administrator. The number of subintervals should then be chosen accordingly.

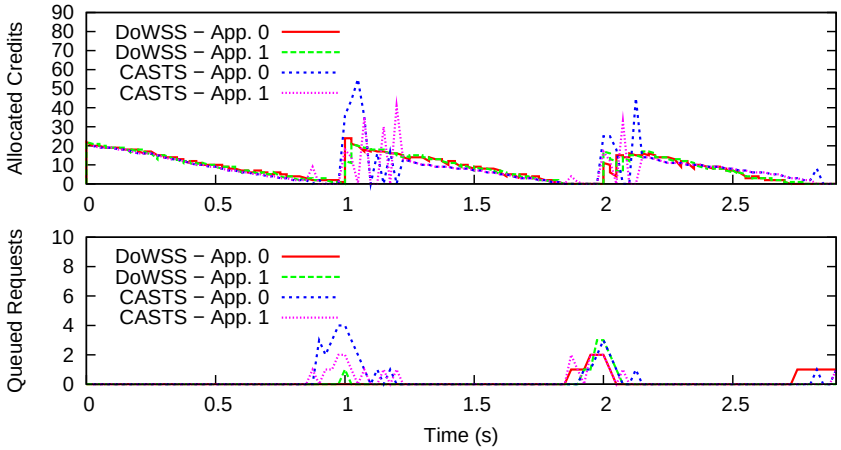


Fig. 9. Number of allocated credits and queued requests on two appliances for $Y_{in} = 128$ reqs/s

1) *Processed Requests* (used in Fig. 6, 7, 10, 11, 12, and 13): Number of requests that were processed by the appliances and sent to the service host.
2) *Allocated Credits* and *Queued Requests* (used in Fig. 9): Respectively, the number of credits that were allocated to a particular appliance and the number of queued requests.
3) *Accumulated Unprocessed Requests* (used in Fig. 8): Number of unprocessed requests per enforcement period among all the appliances accumulated over time. Note that, as stated in Definition 1, the optimal algorithm should lead to $\min[X \times T; Y_{in} \times T]$ and therefore have zero unprocessed requests.

5.3 Performance under Uniform Traffic

In Fig. 6 the performance of DoWSS over one observation period is depicted, under the assumption of uniform traffic. This figure shows the number of processed requests during three enforcement periods. In Fig. 6(a), Y_{in} is set to 128 requests/s. At the end of the observation period, using DoWSS and CASTS the system has processed more requests than MSA. Nevertheless, the value of processed requests never reaches $X \times T$, represented by a horizontal dotted line. This is due to the actual number of requests sent to the appliances. Since the data points represent averages over all the conducted simulations, in some cases Y_{in} might be less than $X \times T$. In Fig. 6(b), Y_{in} is set to 1,000 requests/s. In this case the contract limit X, represented by a horizontal dotted line, is achieved very early in the observation period. When using DoWSS the system performs optimally by processing exactly $X \times T$ credits.

In Fig. 7 we explore the performance of DoWSS during an enforcement period under uniform traffic. This figure shows the number of processed requests during one enforcement period, as a function of the input rate. The horizontal dotted line shows the value of $X \times T$. For sending rates much lower than X, both DoWSS and CASTS perform equally, as expected, while MSA shows a lower performance. However, for sending rates

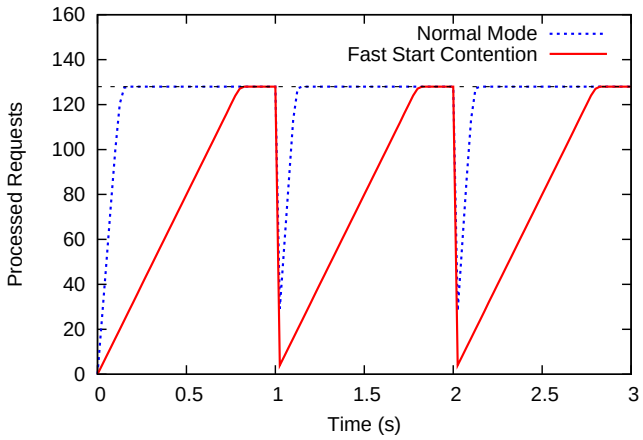
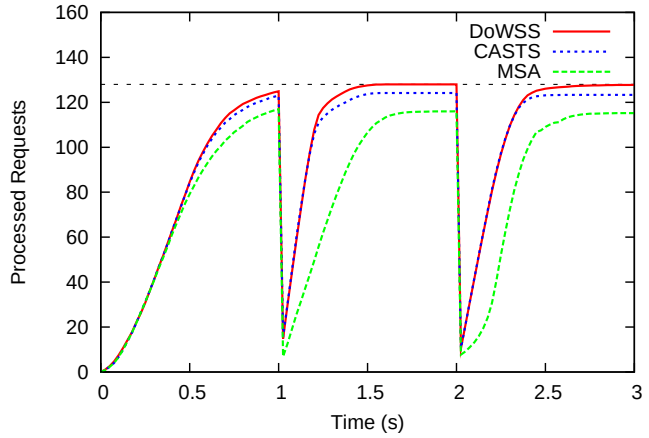


Fig. 10. Performance of the fast start contention mechanism under uniform traffic.

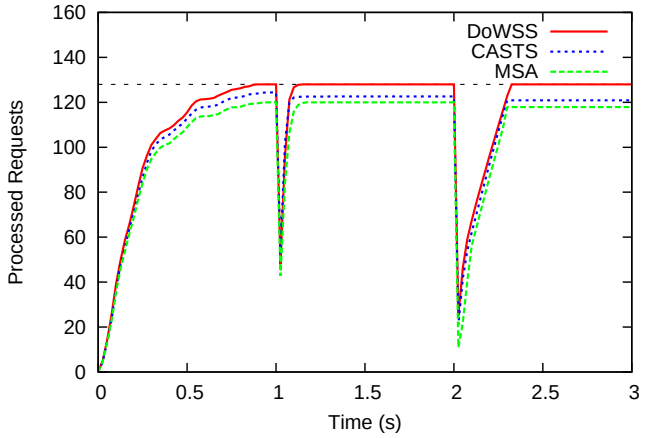
that are close to and above X, DoWSS outperforms both CASTS and MSA. Indeed, while CASTS obtains a good performance by processing a number of requests close to X, DoWSS performs optimally by processing and sending exactly X requests per observation period to the service host.

As mentioned before, the flooring function used in CASTS and MSA will limit in most cases the number of allocated credits to be less than $X \times T$, depending on the number of appliances used. Therefore, a number of requests are left unprocessed at the end of each enforcement period. *These requests accumulate over time, having a negative impact on the overall performance of the system.* Fig. 8 shows the impact of the flooring effect over time. After six hours running the three algorithms, DoWSS has processed up to 170,000 more requests than MSA, and around 120,000 more than CASTS. Clearly, by using DoWSS, the system exploits its maximum capacity.

To illustrate the starvation problem, Fig. 9 shows the number of allocated credits and queued requests for only two of the 10 used appliances. When using CASTS, during the first observation period, as there are no queued



(a) $Y_{in} = 128$ reqs/s.



(b) $Y_{in} = 1000$ reqs/s.

Fig. 11. Number of processed requests per enforcement period (T) for $Y_{in} = X$ and $Y_{in} >> X$ under bursty traffic.

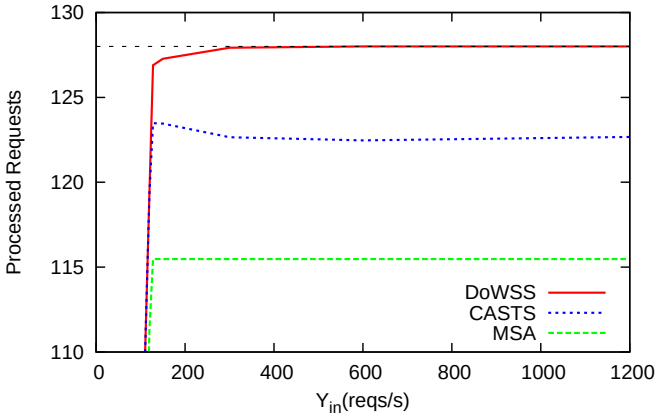


Fig. 12. Total number of sent requests for different input rates under bursty traffic.

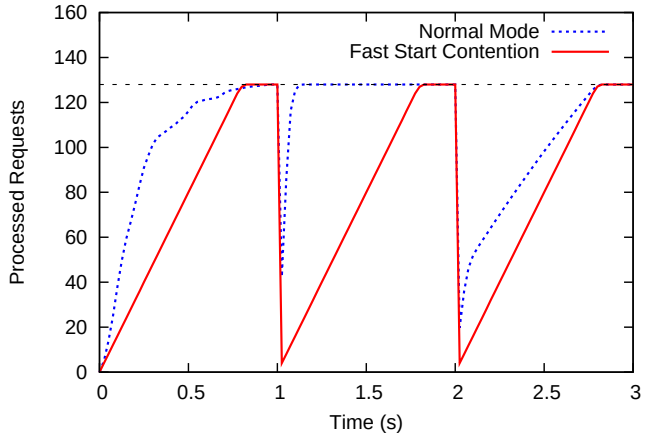


Fig. 13. Performance of the fast start contention mechanism under bursty traffic.

requests, the credits are uniformly distributed among all the appliances. During the subsequent observation periods, when there are queued requests, the appliance with the largest number of requests in queue will obtain most of the assigned credits. However, when there is only one appliance with queued requests, as is the case of appliance 1 around 1.25 seconds, it will claim all the credits during the subperiod. The other appliances are then left creditless, even with empty queues, and are therefore unable to process requests during the subperiod. DoWSS corrects this issue. After 1 second, appliance 1 has queued requests, while appliance 0 has an empty queue. Even with an empty queue, appliance 0 is allocated credits, and thus it is able to process any requests that arrive during the subperiod.

In Fig. 10, we illustrate the performance of the fast start contention mechanism under uniform traffic. Since the fast start depends directly on the number of SON Appliances used (cf. Section 3.3.3), for this set of simulations we set $B = 4$ and $Y_{in} = 1,000$ reqs/sec. If the fast start contention method is not used, the available credits are rapidly consumed during the first few enforcement subperiods, therefore sending the entire number of requests allowed by the SAR during a short period of time. This situation could result in effectively overwhelming the service host. When the contention method is used, the credits are consumed during the entire duration of the enforcement period. In this way, DoWSS further prevents the service host from being overwhelmed.

5.4 Performance Under Bursty Traffic

In Fig. 11, the typical performance of DoWSS under bursty traffic is observed. This figure shows the number of processed requests during three observation periods. In Fig. 11(a), Y_{in} is set to 128 requests/s. Unlike previous performed simulations with uniform entry traffic, under bursty traffic the value of allocated credits reaches $X \times T$, represented by a horizontal dotted line, rather early in the enforcement period. This is due to the actual entry rate to the system. Since the requests are sent in bursts,

the number of requests sent to the SON appliances may reach the maximum specified value at any point in the enforcement period. At the end of the first observation period, DoWSS and CASTS have allocated more credits than MSA. In Fig. 11(b), Y_{in} is set to 1,000 requests/s. In this case, the performance of DoWSS is similar to the performance under uniform traffic. For both entry rates, the performance of CASTS and MSA deteriorates over time.

In Fig. 12, we explore the performance of DoWSS as a function of the input (bursty) traffic. This figure shows the number of processed requests during one enforcement period. The horizontal dotted line shows the value of $X \times T$. Even with bursty traffic, for sending rates much lower than X, both DoWSS and CASTS perform equally and as expected, while MSA shows a lower performance. However, for sending rates that are close to and above X, DoWSS outperforms both CASTS and MSA. Indeed, CASTS obtains a good performance by processing a number of requests close to X, for input values close to X. However, the performance of CASTS decreases as Y_{in} increases. On the other hand, DoWSS performs optimally by processing X requests per observation period.

In Fig. 13, we illustrate the performance of the fast start contention mechanism under bursty traffic. For this set of simulations, we set $B = 4$ and $Y_{in} = 1,000$ reqs/sec. If the fast start contention method is not used, after a short period of time without any received requests, the available credits are rapidly consumed during the few subsequent enforcement subperiods and therefore a substantial number of requests is sent to the service host during a small period of time. When the contention method is used, credit consumption is distributed almost homogeneously over the duration of the enforcement period, thus further preventing the service host from being overwhelmed.

6 Discussion

6.1 Optimality of Results

The obtained results show that the performance of both CASTS and MSA, although satisfactory, is below the performance of DoWSS. The difference is significant as it leads to increasing cumulative underperformance in the former cases. Indeed, since DoWSS complies with $R(K) = \min[X \times T; Y_{in} \times T]$, it has an optimal performance. The flooring function used in both existing credit-based approaches will limit in most cases the number of allocated credits to be less than $X \times T$, which is the maximum allowed by the CSC, depending on the number of appliances used. DoWSS will achieve this maximum while respecting the CSC. It is worth noting again that, even though the numerical difference between the results obtained using DoWSS and those obtained using CASTS is not large, the optimal performance of DoWSS has a substantial positive impact on the performance of the system over time.

A direct consequence of the suboptimal performance of existing credit-based approaches, is the great amount of unprocessed requests which accumulates over time. Given the costs of implementing SON and issues inherent to the provision of Web Services, it is imperative to design efficient algorithms that optimize the overall utilization of the system. As DoWSS provides "optimal" performance, the system is able to exploit its maximum capacity. Therefore, the implementation of DoWSS maximizes the benefits of implementing a two-tiered architecture as the one considered in this paper.

By avoiding the *starvation* issue, the algorithm prevents appliances from going idle in instances where their resources should be exploited. In other words, even with an empty queue, an appliance is allocated credits, and thus it is able to process any requests that arrive just after credit allocation has been performed. This further allows to use the system's resources to their maximum capacity.

The *fast start* contention method included in DoWSS service further enhances protection of the service host. Nevertheless, by using this optional method, another manual rate assignment is being performed, and therefore the desired weighted effect is almost lost. However, it is worth noting that, even though *fast start* is an undesired phenomenon, it is nonetheless inherent to the way credit-based algorithms work.

6.2 Communication Overhead

The proposed algorithm depends crucially on the information exchange between SON appliances in order to take decisions locally at each SON appliance. The amount of information exchanged during each enforcement period will depend directly on the number of enforcement subperiods defined. The goal of dividing the enforcement period into subperiods is to give the algorithm more chances to react to changes in the behavior of the input conditions. Nevertheless, the number of enforcement subperiods has a great impact on the behavior of the algorithm, since the higher the number of subintervals, the higher the control overhead.

The number of control messages exchanged is linearly proportional to both the number of appliances and the number of subintervals. Moreover, the duration of the enforcement period determines the optimum value of the number of enforcement subperiods. Indeed, a larger enforcement period could support a large number of subperiods. Consequently, depending on the length of the observation period and on the size of the requests, the control overhead might compromise the efficiency of the system.

6.3 Open Issues

Scalability. This is a fundamental issue in the design and operation of an enterprise network. The scalability of DoWSS depends on a number of factors such as the SAR, the length of the enforcement period, and

the number of enforcement subperiods. To this end, enterprise networks will need to implement metrics for the scalability of DoWSS. In this work we deliberately did not conduct a study on the scalability of DoWSS. Instead we focused on the core functionalities of our algorithm.

Network Latency. In this body of work, we focused on studying the performance of our approach in single-site data centers, where the network that connects the appliance and server tiers has very small latency. Further study is therefore required for geographically distributed data centers, as the network may now introduce a higher latency.

Distributed Shaping. In this paper we have considered a decentralized deployment architecture in which there are several SON appliances accessing a single service host. In this scenario, each appliance runs the proposed algorithm while having a **global** knowledge of the state of the rest of the appliances. To this end, the appliances exchange information with each other. Each appliance calculates locally the sending rate. Nevertheless, there is another deployment scenario where the algorithm is completely distributed, and each appliance will calculate locally the sending rate towards the service host while having a **partial** knowledge of the state of the rest of the appliances. In this paper we did not consider this scenario, and focused on the decentralized case.

CSC Definitions. One of the major issues of implementing Service Traffic Shaping algorithms in Service-Oriented Networks, is the fact that the Client Service Contracts are ill-defined. Indeed, CSCs are defined in terms of requests per second as opposed to data units per second as it is done in classic networking. Moreover, they do not include parameters such as a maximum burst size. Consequently, CSCs in SON are not precisely measurable, making the resource (CPU processing time in this case) difficult to protect. The challenge is therefore, finding a way to establish other types of CSCs that are well-defined.

Real Testbed. The simulation results obtained by this work show the strengths and usefulness of the approach, as well as some of its limitations and possible drawbacks. The next step of the work is to design and implement a practical version on a real world testbed in order to properly measure the impact of the algorithm in an actual production environment.

7 Related Work

Many research efforts in the literature are oriented specifically towards QoS control in Web server farms. Nevertheless, these efforts center around enforcing SLAs defined in terms of maximum response times of a service. To this end, most of the methods are based on either service differentiation and prioritization [18], [19], or admission control of new service requests [20], [21], or both [22], [23].

In particular, the work that is the closest to ours is the one of Garcia et al. [24]. The authors analyze the deficiencies in existing methods, define a number of requirements that any QoS control mechanism working in an enterprise environment should address, and establish their basic design principles. Along these lines, the authors propose a QoS control mechanism which determines the maximum number of concurrent sessions that a service host can process, and calculates the maximum admissible sessions in order to maintain the values specified by the SLA.

It is worth noting that, most of these research works involve the use of a centralized server and measurement/processing at the service hosts. Since the idea behind using a multi-tier architecture is to offload some tasks to be done from the servers, these methods are not applicable in our particular context. In contrast, our work aims at a decentralized method for enforcing the CSC. Furthermore, we do not center our work on service response times or client differentiation and scheduling. Instead, our objective is to prevent service hosts from being overwhelmed.

8 Conclusions

Service-Oriented Architectures (SOA) have emerged as the main solution for the integration of legacy systems with new technologies in the enterprise world. Services are governed by client service contracts (CSCs) that specify, among other requirements, the rate at which a service should be accessed in terms of "X requests" per observation period T, in order to prevent a service host from being unduly overwhelmed. Several approaches, using both static and dynamic, credit-based strategies, have been developed in order to enforce the access rate specified in such a contract. Nevertheless, because client service contracts are ill-defined (e.g., due to lack of parameters such as a maximum burst size), existing approaches have problems related to starvation, approximations used in calculations, and rapid credit consumption under certain conditions.

In this paper we presented DoWSS, a doubly-weighted algorithm for service traffic shaping in service-oriented networks. Contrary to existing credit-based approaches, our approach guarantees the allocation of at least one credit per measurement subperiod, thus effectively solving the numerical approximation issues, by exploring the communication capabilities of the SON Appliances. DoWSS involves the use of a doubly-weighted strategy for credit allocation, using weights based on request sizes. Therefore, DoWSS effectively penalizes the appliance queues that would take the longest to process, by assigning more credits to appliances with smaller queues, thus preventing *starvation*. The algorithm also introduces a procedure to contain the rapid consumption of credits at the beginning of enforcement period, or *fast start*, further preventing the service host from being overwhelmed.

We evaluated the performance of DoWSS by conducting a series of simulations. The obtained results show that DoWSS performs optimally by processing exactly $X \times T$ requests per observation period, which is the maximum possible number of requests allowed by the client service contract. We also show that our approach has a substantial positive impact over time on the overall performance of the system, by using it to its maximum capacity.

REFERENCES

[1] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. (2004, February) Web Services Architecture. [Online]. Available: http://www.w3.org/TR/ws-arch/

[2] W3C, "Web services description language (wsdl) 1.1." [Online]. Available: http://www.w3.org/TR/wsdl

[3] ——, "SOAP Version 1.2 Part 0: Primer (Second Edition)." [Online]. Available: http://www.w3.org/TR/2007/REC-soap12-part0-20070427/

[4] OASIS, "Uddi version 3.0.2." [Online]. Available: http://uddi.org/pubs/uddi_v3.htm

[5] S. Zilora and S. Ketha, "Think inside the box! optimizing web services performance today," *IEEE Communications Magazine*, vol. 46, no. 3, pp. 112 – 117, March 2008.

[6] M. Head, M. Govindaraju, R. Engelen, and W. Zhang, "Benchmarking XML processors for applications in grid web services," in *ACM/IEEE Conference on Supercomputing*, Tampa, FL, USA, November 2006.

[7] R. D. Callaway, A. Rodriguez, M. Devetsikiotis, and G. Cuomo, "Challenges in service-oriented networking," in *IEEE Globecom*, San Francisco, CA, USA, November 2006.

[8] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single-node case," *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 344 – 357, June 1993.

[9] A. Elwalid and D. Mitra, "Traffic shaping at a network node: theory, optimum design, admission control," in *IEEE Infocom*, Kobe, Japan, March 1997.

[10] B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, and A. Snoeren, "Cloud Control with Distributed Rate Limiting," in *ACM Sigcomm*, Kyoto, Japan, August 2007.

[11] K. Boloor, M. Dias de Amorim, B. Callaway, A. Rodriguez, and Y. Viniotis, "Evaluation of multi-point to single-point service traffic shaping in an enterprise network," in *IEEE Globecom*, Honolulu, HI, USA, December 2009.

[12] A. Tanenbaum, *Modern Operating Systems*. Prentice Hall, 2001.

[13] V. Muthusamy and H.-A. Jacobsen, "SLA-Driven Distributed Application Development," in *MW4SOC*, Leuven, Belgium, December 2008.

[14] G. Cuomo, "IBM SOA "on the edge"," in *ACM Sigmod*, Baltimore, MD, USA, June 2005.

[15] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *Computer Communications Review*, vol. 19, pp. 1–12, August 1989.

[16] A. Varga, "The OMNeT++ Discrete Event Simulation System," in *European Simulation Multiconference*, Prague, Czech Republic, June 2001.

[17] M. Zukerman, T. Neame, and R. Addie, "Internet traffic modeling and future technology implications," in *IEEE Infocom*, San Francisco, CA, USA, January 2003.

[18] J. Wei, X. Zhou, and C.-Z. Xu, "Robust processing rate allocation for proportional slowdown differentiation on internet servers," *IEEE Transactions on Computers*, vol. 54, no. 8, pp. 964–977, August 2005.

[19] B. Schroeder and M. Harchol-Balter, "Web servers under overload: How scheduling can help," *ACM Transactions on Internet Technology*, vol. 6, no. 1, pp. 20–52, February 2006.

[20] L. Cherkasova and P. Phaal, "Session-based admission control: a mechanism for improving performance of commercial Web sites," in *IEEE International Workshop on Quality of Service*, London, England, Jan 1999, pp. 226 – 235.

[21] X. Chen, P. Mohapatra, and H. Chen, "An admission control scheme for predictable server response time for web accesses," in *International World Wide Web Conference*, Hong Kong, Hong Kong, January 2001.

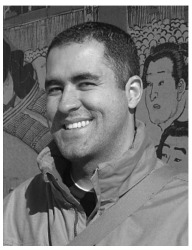
[22] N. Bhatti and R. Friedrich, "Web server support for tiered services," *IEEE Network*, vol. 13, no. 5, pp. 64 – 71, September 1999.

[23] K. Li and S. Jamin, "A measurement-based admission-controlled Web server," in *IEEE Infocom*, vol. 2, Tel-Aviv, Israel, March 2000.

[24] D. Garcia, J. Garcia, J. Entrialgo, M. Garcia, P. Valledor, R. Garcia, and A. Campos, "A QoS Control Mechanism to Provide Service Differentiation and Overload Protection to Internet Scalable Servers," *IEEE Transactions on Services Computing*, vol. 2, no. 1, pp. 3 – 16, January 2009.

Yesid Jarma received the B.S. Degree in Electronic Engineering from Universidad Distrital "Francisco José de Caldas" in Bogotá, D.C., Colombia, in 2005. He obtained the M.Sc. degree in Computer Science from UPMC Sorbonne Universités in Paris, France, in 2008. He's currently a final year Ph.D. student at the computer science laboratory (LIP6) of UPMC Sorbonne Universités. His research interests are service-oriented networking and wireless self-organizing networks.

Keerthana Boloor Keerthana Boloor is currently a Ph.D. student in the Department of Electrical and Computer Engineering at North Carolina State University, Raleigh, NC. Her Ph.D. research is on building Service Level Agreement (SLA)-based management policies for SOA applications deployed in a distributed cloud. She is a two-time IBM Fellowship award winner. Her research interests are in the fields of service oriented computing and distributed systems.



Marcelo Dias de Amorim is a CNRS permanent researcher at the computer science laboratory (LIP6) of UPMC Sorbonne Universités, France. His research interests focus on the design and evaluation of dynamic networks as well as service-oriented architectures. For more information, visit http://www-npa.lip6.fr/~amorim

Yannis Viniotis Yannis Viniotis received his Ph.D. from the University of Maryland, College Park, in 1988 and is currently Professor, Department of Electrical and Computer Engineering at North Carolina State University. His research interests include Service-Oriented Architectures, service engineering, and design and analysis of stochastic algorithms. Dr. Viniotis was the cofounder of Orologic, a successful startup networking company in Research Triangle Park, NC, that specialized in ASIC implementation of integrated traffic management solutions for high-speed networks.

Robert D. Callaway received B.S. degrees in Computer and Electrical Engineering, the M.Sc. degree in Computer Networking, and the Ph.D. degree in Computer Engineering from North Carolina State University in 2003, 2004, and 2008 respectively. Dr. Callaway is currently a Senior Software Engineer with IBM, as well as an Adjunct Assistant Professor at NC State University. His research and development interests are in service engineering, application-aware networking, and middleware. He was the recipient of a 2007-2008 IBM PhD Fellowship.