Journal of Network and Systems Management, Vol. 15, No. 2, June 2007 (© 2007)
DOI: 10.1007/s10922-007-9059-8

Performance Analysis of Adaptive Admission Control Algorithms for Bandwidth Brokers

Ch. Bouras[1,2] and K. Stamos[1]

Published online: 24 May 2007

In this paper, we propose a novel architecture for the admission control module of a Bandwidth Broker entity that aims at achieving a satisfactory balance between maximizing the resource utilization for the managed network and minimizing the overhead of the module. We also describe, analyze and evaluate mechanisms which aim at solving the additional problems of fairly prioritizing resubmitted requests and efficiently handling requests which do not specify ending times. We use the well known network simulator ns-2, as well as a custom simulation environment in order to study the performance characteristics of the proposed mechanisms and compare them with various alternatives for the admission control module of a Bandwidth Broker. We provide the results of the experimental evaluations and the conclusions they lead us to for the relative importance of the proposed solution and the various alternatives, their advantages and drawbacks, and the environments for which each one is best suited.

KEY WORDS: Bandwidth broker; admission control; DiffServ; adaptive; SLA.

1. INTRODUCTION

Bandwidth Brokers are entities proposed in the framework of the DiffServ architecture for Quality of Service (QoS) provision in the Internet. In order to offer better scalability than other architectures such as IntServ, the DiffServ architecture [1] only deals with individual flows at the edges of a domain, allowing the core elements of the network to only handle classes of service. Within this framework, a Bandwidth Broker [2] is an entity responsible for providing QoS within a network domain. The Bandwidth Broker manages the resources within the specific domain by controlling the network load and by accepting or rejecting bandwidth requests. Every user (service operator) who is willing to use an amount of the network

[1]Research Academic Computer Technology Institute, Greece and Computer Engineering and Informatics Dept., Univ. of Patras, PO Box 1122, GR-26500 Patras, Greece.
[2] To whom correspondence should be addressed at. E-mail: bouras@cti.gr, stamos@cti.gr.

resources, between its node and a destination, sends a request to the Bandwidth Broker. The choice of the Bandwidth Broker to either accept or reject a request is based on the network load and on the Service Level Agreement (SLA). For requests that span multiple domains (inter-domain requests), the Bandwidth Broker will have to communicate with Bandwidth Brokers in the adjacent domains that are traversed by the requested flow. Bandwidth Brokers only need to establish relationships of limited trust with their peers in adjacent domains, unlike schemes that require the setting of flow specifications in routers throughout an end-to-end path. Therefore, the Bandwidth Broker architecture makes it possible to keep state on an administrative domain basis, rather than at every router and the DiffServ architecture makes it possible to confine per flow state to just the leaf routers.

A Bandwidth Broker contains several modules which are necessary for its transparent and efficient operation (Fig. 1). These modules include an inter-domain interface for communication with adjacent Bandwidth Brokers in neighboring domains, an intra-domain interface for communication with the service components that are located inside the domain controlled by the Bandwidth Broker, a routing table interface which is used so that the Bandwidth Broker is aware of the network topology and the routing paths, a user/application interface, a policy manager interface for implementation of complex policy management or admission control, and a network management interface for coordination of network provisioning and monitoring. A diagram that displays the above modules and their interfacing within the Bandwidth Broker is shown in Fig. 1

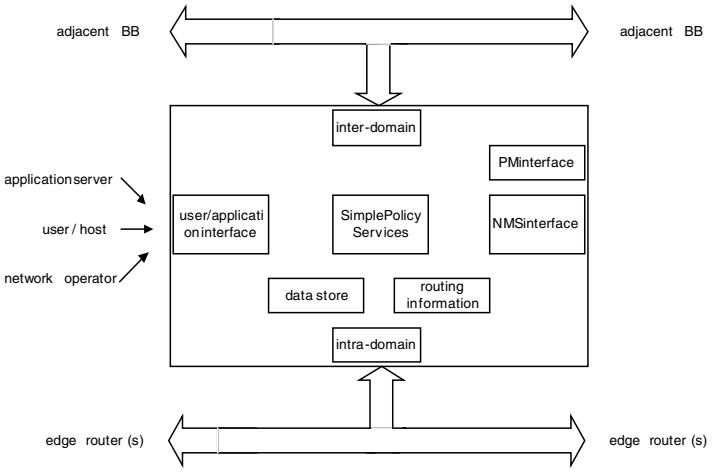

Fig. 1. Bandwidth Broker diagram.

Bandwidth Brokers are an intensely studied field, and a number of architectures have been proposed for the various aspects of its operation ([3], [4], [5], [6]). Admission control means that the Bandwidth Broker has to define whether the incoming resource reservation requests will be accepted or not. Once a request has been accepted, the Bandwidth Broker has to make sure that it will be met by the network. Admission control is a very important part of the Bandwidth Broker operation, because it determines the fairness between the requests and the degree of network utilization that the Bandwidth Broker will achieve for the managed domain. There are various definitions of fairness, which depend on the context used. In this case, we use a loose general definition of fairness as the unnecessary downgrading of the admission control service for the users. An improperly designed admission control module can lead to low network utilization, unfairness and therefore frustration to the users that request resources or it can also impose an unacceptable overhead to the Bandwidth Broker's operation. However, since the operating circumstances can vary widely from one case to another, it is improbable that a single solution will fit all operating requirements. Our approach tries to tackle this problem by incorporating an adaptive mechanism with the intent to converge to the suitable level for each deployment scenario.

In general, we can separate the types of reservation requests depending on the actual time period for which they request resources.

1.1. Immediate Requests

When an immediate request is accepted, it is immediately effective, which means that the requested resources are reserved right away. This type of request leaves little room to the Bandwidth Broker for implementing a strategy that maximizes the network utilization.

1.2. Book-Ahead (Or Advance) Requests

A book-ahead request specifies the resources that will be needed at some later point in time, which has to be specifically defined. The authors in [7] give a thorough presentation of the concept of book-ahead reservations, while a detailed discussion on the benefits and potential problems with book-ahead requests can be found in [8]. In general, book-ahead requests provide a richer functionality to the service and allow for better solutions to the admission control problem. There are a lot of actual cases in the real world where a book-ahead request meets the requirements of an application, like for example pre-arranged video conferences.

The main problem we deal with in this paper is how to better examine and admit incoming requests in a DiffServ-enabled domain that is automatically managed by a Bandwidth Broker or similar entity. The variety of environments, requirements and network conditions that affect such a mechanism make the research

for an efficient solution to this problem important and useful. There are emerging types of networks, such as wireless networks, where the demand for quality of service is strong, and where sophisticated solutions are applicable because of the nature of the wireless environment: smaller capacity links compared to wired networks, frequent and more unpredictable entry of new users for the existing resources. Other research teams have already taken over the work presented in this paper in order to apply it to wireless environments.

In the next sections of this paper, we examine an adaptive admission control module for Bandwidth Brokers and the improvements in the architecture's efficiency that it can lead to. We also examine how the admission control module could take advantage of, or handle, special conditions such as the resubmission of previously rejected requests. We also use simulations in order to make several observations on the behaviour of the algorithm, and compare it with several variations of admission control algorithms.

The rest of the paper is organized as follows: Section 2 presents related work on the issue of admission control in Bandwidth Brokers, while section 3 discusses the adaptive admission control algorithm and section 3.1 presents some ideas on the possible improvements to it. The enhanced algorithm is described and analyzed in section 3.2, while section 3.3 discusses further related issues. Initial performance evaluations of the algorithm are presented in section 4, while section 4.3 presents in detail the implementation and simulations of the described mechanisms in the popular ns-2 simulator. Finally, section 5 summarizes our conclusions and our future work in this area.

2. RELATED WORK

Researchers have dealt with both types of admission requests. An approach that deals with both types of incoming requests is the resource partitioning proposed in [9], which separates the admission decision for immediate and book-ahead requests. Immediate requests that were rejected can be reconsidered for a book-ahead reservation. In order to avoid wasting of resources because of fragmentation, the authors propose a moving boundary between the two partitions. The most common mechanism for admitting book-ahead requests is to divide time in intervals (slots) of equal size, and calculate the resources requested by a new reservation for the time slots that it overlaps [9], [10].

For both immediate and book-ahead requests, it may be possible to either specifically declare the ending time of the reservation, or not. An intermediate case is when a reservation request has to provide both its starting and ending times, but can make new additional requests that extend its initial reservation period. In [11], the authors present a Bandwidth Management Point (BMP) that uses centralized network state maintenance and pipe-based intra-domain resource management schemes in order to reduce the admission control time and the scalability problems.

Another approach is used by [12], where SLA requests are converted by the border routers in messages of the COPS (Common Open Policy Service) protocol and refined before they are sent to the Bandwidth Broker, which makes simple decisions based on the refined requests.

The intra-domain admission control that admits or rejects new flow requests based on the knowledge of available resources within the domain and flow requirements has been studied in [13], [14] and [15]. Reference [16] presents an inter-domain approach to the admission control.

In some cases, a book-ahead request may have a flexibility of allowing the Bandwidth Broker to answer the request by either accepting it or rejecting it not immediately, but after a period of time (which can be specified). Our algorithm takes advantage of this flexibility, in order to calculate more efficient admission decisions that try to achieve better network utilization and therefore better profit for the network provider than simpler techniques. The provider may choose to allow requests that demand an immediate answer, balancing perhaps this capability with additional cost.

Most related work for Bandwidth Brokers examines a request as soon as it arrives and accepts it if the reservation does not exceed the unreserved link capacity [17]. This approach has benefits in terms of speed and efficiency, but it can lead to low network utilization. In [18], the authors show how the general admission control problem can be formulated as an optimization problem, with the goal of maximizing the net revenue. The network utilization can improve drastically if we allow the Bandwidth Broker's admission control to gather a number of requests and compute a better allocation of resources. This is the approach we have taken in this paper. Also [19] deals with price-based admission control, studying both online (when answers to requests have to be issued immediately) and offline (when requests can be gathered and evaluated) versions of the problem are discussed. Our work combines the above approaches with an adaptive scheme that attempts to achieve a preferable balance between optimal utilization of the network and minimal overhead for the Bandwidth Broker operation. By optimal utilization, we define the admission of such a set of requests that is feasible (i.e. maximum allocated bandwidth for the service is not exceeded at any time), and there is maximum utilization of network resources (expressed in transferred bytes). A more formal definition is provided in section 3. By minimal overhead, we define the computational overhead for simply checking whether an incoming request is feasible, by comparing the remaining resources allocated for the service (after subtracting the request reservation) to zero. This approach is described in more detail in section 4.3, upon discussing the Simple Admission Control (SAC) algorithm. An adaptive scheme designed for the scheduling of popular video on demand that also uses delayed notification is presented in [20]. In [21], the admission control approach taken by the TEQUILA project is presented, which is based on a feedback model that can be tuned by operational policies and strategies.

Its main characteristic is that it is based on the ability of the network to provide QoS using functions that dimension the network on the basis of anticipated demand, and on the actual status of the network using measurements. A thorough overview and classification of network admission control techniques is provided in [22]. In the issue of routing bandwidth guaranteed paths in MPLS networks, the authors in [23] introduce a dynamic online algorithm that tries to optimize and balance traffic load in the network.

The TISPAN standardization body of the European Telecommunications Standards Institute (ETSI) [24] has produced the Resource and Admission Control Sub-system (RACS) specification identified in the overall TISPAN Next Generation Networking (NGN) architecture. RACS [25] is the TISPAN NGN subsystem, responsible for elements of policing control including resource reservation and admission control in the access and aggregation networks. Specifically, RACS defines a number of functional elements such as the Service Policy Decision Function (SPDF), which is a logical policy decision element for Service-Based Policy control (SBP) and the Access-Resource and Admission Control Function (A-RACF), which main functions are the admission control and the assembly of network policy. Although therefore the proposed architecture in this paper is structured around the IETF Bandwidth Broker recommendations [2] and the proposals by Qbone [3], it fits into this model by studying the admission control function (part of the A-RACF), as well as the service policy decision function of the SPDF. More specifically, the mechanisms described in this paper may be plugged within a RACS framework in order to implement the functionalities defined by A-RACF and SPDF. SPDF is more closely mapped to the higher level policy functionalities while the A-RACF module can be mapped to lower level functionalities responsible for the admission control.

In the specific area of providing end to end QoS requirements for a PSTN grade voice and multi-media service, the MultiService Forum (MSF) has proposed a QoS solution [26] and considered how it might best be supported over a packet network infrastructure. This solution has a number of components, among which is the Bandwidth Manager [27], and has chosen to adopt the DiffServ PIB (Policy Information Base) for the interface between the Bandwidth Manager and the edge routers of a domain. It covers hierarchical organization of Bandwidth Manager components, and reservations between managers in adjacent domains. This Bandwidth Manager has the basic characteristics of the Bandwidth Broker component discussed in this paper, and the proposed mechanisms can potentially be integrated in the Bandwidth Manager structure.

Additionally, an end to end QoS solution has also been defined by 3GPP [28], and it incorporates a Policy Decision Function that approximates part of the Bandwidth Broker functionality discussed here. The 3GPP proposal is mainly specialized on 3G mobile networks and defines its own PIB for interfacing to the Policy Decision Function.

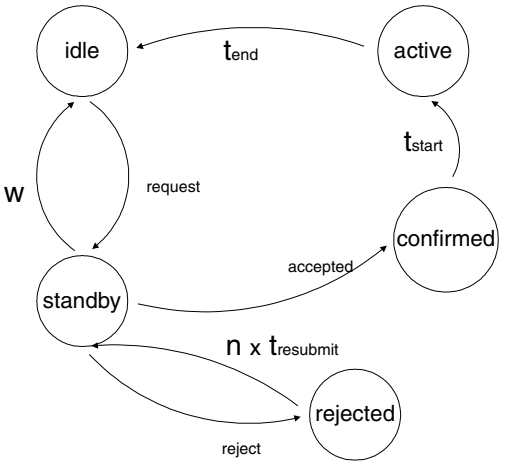

Fig. 2. Request states.

3. ADAPTIVE ADMISSION CONTROL ALGORITHM

We define standby requests as requests that have not yet received an answer (either confirmation or rejection). Confirmed is a book-ahead request that has received an affirmative answer but waits to be activated. These states are shown in Fig. 2.

For a DiffServ domain, it is not efficient to keep per-flow status in the core routing devices, and therefore an aggregated approach is used at the core routers. Admission control is performed by the Bandwidth Broker at the domain scale, using the hose model [29] that has been proposed for VPN provisioning. Its basic idea is that bandwidth management is simplified by assigning a limit at the bandwidth that each edge router is allowed to accept in the domain. Its operation assumes that proper dimensioning of the network has taken place in order to be able to support the hoses definition and that part of the available bandwidth for the links has been assigned to the management of the Bandwidth Broker for the DiffServ service. We have to note here, that this does not mean that all or most of the reservation requests will be necessarily accepted, but that the network design will guarantee that there will be enough resources in order to service the reservation requests that have been accepted. The hose model has the benefit of offering the flexibility to the edge device of sending traffic to a set of endpoints without having to specify the detailed traffic matrix and can also reduce the required size of access links through multiplexing gains because of the aggregation of flows between endpoints. Various methods of implementing the hose service model are given in [29].

An important problem for admission control and dimensioning of network resources is how to take into account the burstiness and self-similar behavior of network traffic. The problem can be simplified by defining a unifying parameter for the characteristics of network resources that accurately captures their effect on resource usage. Various researchers ([30], [31], [32]) have dealt with the concept of effective bandwidth, which can be computed based on traffic parameters such as mean rate or maximum burst size. It is an additive amount of bandwidth that is large enough in order to guarantee that the QoS requirements of all flows are met.

Our approach also decouples the admission control decision from the routing issues within the domain. This means that core routers are not involved in the process of admission control and signalling. Furthermore, while combining routing decisions with admission control can be beneficial, it is not always possible, or desirable for scalability reasons. Packets of a specific flow can be routed using different paths, without affecting the admission control process.

We suppose a new request has the form of

$$r(t_{\text{start}}, t_{\text{end}}, b, w) \tag{1}$$

where t_{start} and t_{end} are the starting and finishing times for the reservation, b is the requested bandwidth and w is the period for which the request can wait until it receives either a confirmation or a rejection of the reservation. Our model is based the peak rate allocation, so that each request specifies its maximum transmission rate and the admission control module has to make sure that the sum of all peak rate does not exceed the allocated service capacity. Bandwidth is defined at Layer 3, as the maximum amount of bits/second of IP traffic, including the IP header that an end point may insert to the network. The Bandwidth Broker's admission control module keeps a list of unanswered requests, which we call waiting queue W_q, sorted by their waiting time w.

As soon as the first item, say r_1 (with the closest w to the current time) is about to expire, the admission control module calculates the answers that it will provide to this and a number of other requests, essentially by solving an offline scheduling problem:

Suppose n is the cardinality of W_q. We define

$$R = \{r_1, r_2, \cdots, r_m\} \subseteq W_q \tag{2}$$

and we want to find a subset $R_c \subseteq R$ such that $\sum_{ri \in R_c} b_i \leq B$ at any time point where B is the total available bandwidth for the service and try to maximize $\sum_{ri \in R_c} b_i$ throughout the period from the earliest t_{start} to the latest t_{end} in the R set. R_c is the set of requests that will be accepted by the algorithm, while requests in the set R-R_c will be rejected.

This problem is NP-complete [33] and therefore it is proposed to use an approximation algorithm to solve the following linear programming relaxation in polynomial time [19] and then make the solution discrete regarding the variables

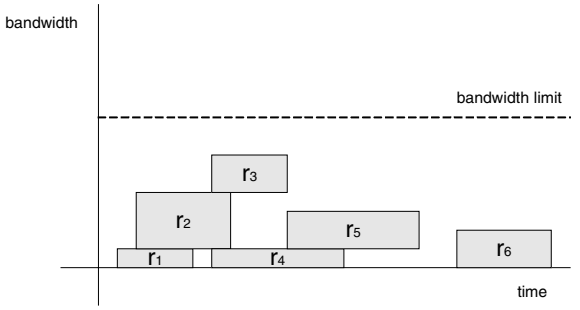

Fig. 3. Reservation requests.

x_i, which represent whether r_i is accepted or not:

$$\max \Sigma_{i \in R} b_i (t_{end} - t_{start}) X_i$$

$$\Sigma_{i \in R(t)} b_i x_i \leq B, \text{ for all } t \in (\text{earliest } t_{start}, \text{ latest } t_{end}) \text{in } R$$

$$0 \leq x_i \leq 1, i \in R$$

In order to avoid approximation of the solution, this problem can be solved using integer linear programming, which however becomes very costly computationally as the problem instance increases (which happens for a high rate of incoming requests). A mechanism monitoring the instance size and carefully adapting it is needed in this case. The important point is how to select the R set. A simple approach would be to simply set $R = W_q$. This solution however can become computationally costly, and it can furthermore lead to low network utilization, because requests that have been made very far in advance will probably have little competition, and will therefore be most likely accepted. In the example in Fig. 3, this can be demonstrated by request r_6. If r_6 is included in the R set as soon as r_1, it will certainly be accepted, since there it has no other competition. It would be better though to delay the decision for r_6, since by that time other requests (more profitable than r_6) could have arrived.

Including in R only requests that overlap with t_{end} for r_1 may also not be an attractive solution, because it might require the algorithm to be invoked too frequently, and that could introduce an unacceptable overhead. In Fig. 3, the algorithm will have to separately be invoked for r_1, r_3, r_5 and r_6. As our algorithm does not provide for overbookings, we have also drawn a line that shows the maximum bandwidth that can be allocated to the requests.

In order to combine the benefits of both these extremes and reduce their shortcomings, our solution is to have an adaptive parameter for the size of R, which will increase if the number of requests in W_q increases or if the algorithm

was very time-consuming, and decrease otherwise, according to:

$$\text{if}(C_{k-1} - T > T)$$

$$R_{size}(k) = 1$$

$$\text{elseif}(C_{k-1} - T > 0)$$

$$R_{size}(k) = (1 - (C_{k-1} - T)/T)R_{size}(k - 1)$$

$$\text{else}$$

$$R_{size}(k) = R_{size}(k - 1) + (Wq - R_{size}(k - 1))^{*}a$$

C_{k-1} is the duration of the previous computation of the requests to be accepted, a is a parameter that determines the increase rate of R_{size} and T is a threshold value of the maximum allowable time for a computation. Configuring parameter a determines how close to W_q we want the size of the R set to become after an increase, so a is essentially the adaptation factor of the algorithm's operation. The above pseudocode guarantees that if the algorithm lasts more than an accepted threshold, it will be simplified to admission control for a single request, which is the simplest computation possible and we can reasonably expect to reduce the computational overhead to very low levels, from which the algorithm can slowly progress to more complex computations according to the adaptation parameter a. In general, an adaptive scheme can have problems of oscillation between extreme values, so we have considered the above scheme that does not sharply increase or decrease the size of the R set, except for the case that for some reason the computation time far exceeds the acceptable threshold.

As an example, if the last computation lasted more than twice the predefined threshold, then the size of the subset R that will be examined for the current computation is reduced to 1, and therefore the computation is simplified to examine whether accepting the next request violates the resource restrictions or not. The assumption is that in that case the computation of an optimal solution is adding a large overhead to the Bandwidth Broker's operation, and we therefore simplify the computation as much as possible. In the case that the computation time exceeds the threshold but not twice its value, we assume that the overhead is significant and must be reduced (but is not unacceptable as in the previous case). We reduce it by a factor of $1\text{-}(C_{k-1}\text{-}T)/T$, so that the reduction becomes more aggressive as C_{k-1} becomes larger. Finally, if the computation time is still below the threshold, we assume that there is space for increasing the computation overhead by increasing the size of the subset R, and this is done using a factor $a \in (0,1)$. The closest to 1 this factor is chosen, the more aggressive the increase is, with the obvious limit of the size of the whole W_q.

In general, finding an optimal solution to the problem of optimally scheduling the requests is NP-complete, since the Knapsack problem, which is known to be

NP-complete [33], is equivalent to a simpler version of our scheduling problem where all t_{start} and t_{end} times are equal. Because however it is not critical to have the optimal solution, we can either use an approximation scheme that runs in polynomial time and approximates the optimal solution with the desired accuracy, or try to limit the instance size of the problem in order to be within the computational capabilities of the underlying equipment.

Following is a summary of the algorithm for calculating the accepted requests at an ingress point of the domain:

```
while (W_q not empty)
  while (next request has not expired)
    forall i where (b_i > B)
      x_i = 0    // reject overbooking requests
    Solve LP maximization:
    max Σ_{i∈R} b_i(t_end-t_start)x_i
    Σ_{i∈R(t)} b_i x_i ≤ B, forall t ∈ (earliest t_start, latest t_end) in R
    x_i ∈ {0,1}, i∈R
    exit loop
  end while
  if ((C-T)>=T)
    R_size = 1
  else if ((C-T)>0)
    R_size = (1-(C-T)/T)* R_size
  else
    R_size = R_size + (W_q-R_size)*a
end while
```

If the current computation takes too long and w_1 is about to expire, the computation is ignored and a simpler fall-back mechanism is used which will individually examine r_1 and then restart the computation for R-$\{r_1\}$.

Because of the way the algorithm is constructed, it is not generally optimal on network utilization. As we have mentioned, we make this trade-off in order to reduce the computation overhead for the Bandwidth Broker module. A very fast processing module (or conversely a low rate of admission requests) lead the algorithm to quickly converge to the best approximation of the optimal solution.

Thus, assuming that the computation time does not reach or exceed the threshold, we have that

$$R_{\text{size}}(t) = R_{\text{size}}(t-1) + (Wq - R_{\text{size}}(t-1))^{*}a$$

Solving the recursive function gives

$$R_{\text{size}}(t) = (1-a)^{t-1} R_{\text{size}}init + (1 + (1-a) + \ldots + (1-a)^{t-2}) Wq^{*}a$$

and therefore

$$R_{\text{size}}(t) = (1-a)^{t-1} R_{\text{size}}init + (1 + (1-a)\frac{1-(1-a)^{t-2}}{a}) Wq^{*}a \qquad (3)$$

where $R_{\text{size}}init$ is the initial size of R.

Therefore, R_{size} converges to the size of W_q as quickly as $(1-a)^t$ converges to near-zero values, which happens quite rapidly, especially if a has been chosen close to 1, which means a very high adaptation capability.

The algorithm is also not "fair" with respect to maintaining a First Come-First Served order (since an earlier request might be rejected in favour of a later request that will better utilize the network), but it achieves "fairness" with respect to the response to requests (it assures that all requests will be answered on time, either positively or negatively) and it respects each request's maximum waiting time w.

Finally, it has to be noted that there is the assumption that a client will submit requests to the service in a well-behaved way. This may be a valid assumption in small or research environments, but it may not be valid in a larger scale, or in an environment with serious security considerations. A possible solution is to restrict access to the submission service only to trusted / authorized users, using for example an Access List. Also more sophisticated architectures may be necessary ([34], [35], [36]).

3.1. Increasing the Adaptive Admission Control Module Versatility

The algorithm described in the previous paragraph can be improved in several aspects, depending on the implementation environment and the specific request behavior.

- Request resubmission: Requests that have been rejected are notified of a later time when they will have better success chances. Moreover, they are given a chance of reducing their reservation requests, in hope that a satisfactory compromise can be found that will cover the user's needs. This can be achieved by keeping a tentative list of the total bandwidth requested at any time, for both admitted and pending requests.
- Open requests: Some requests may not specify their ending time. The Bandwidth Broker algorithm can be extended to take into account such requests, which we call "open" requests. In order to determine the duration of the reservation, one option for the Bandwidth Broker is to make the conservative assumption that the requested bandwidth will be reserved indefinitely (which means that the bandwidth will only be released upon notification by the party that initiated the request that it no longer needs it). This flexibility however is very costly and the network utilization can be decreased significantly, and therefore can only be offered at a significantly higher cost. A better option is for the Bandwidth Broker to observe the distribution of the existing requests and the durations they request. It can then service open requests with the assumption that their ending time is such that it exceeds the duration of the largest percentage of the non-open requests, thereby reducing the ratio of prematurely interrupted reservations to a minimal percentage

Despite extensive literature in the area of admission control for Bandwidth Brokers, the issue of resubmission of rejected requests has not been as thoroughly investigated. Most researchers implicitly assume that the end user will decide whether a rejected request will be resubmitted as a new request (without the admission control module recognizing that there is a relationship with one or more previously rejected requests). However, identifying such requests and handling them in a special way bears a number of advantages for the overall performance, as we show in later sections of this paper.

3.2. Algorithm Enhancements

For resubmissions of previously rejected requests, there is no change in the user client's request format, which means that by examining just the request, a resubmitted request can not be differentiated from a newly generated one. It is proposed however, that for purposes of avoiding excessive unnecessary overhead, the clients will obey a specified protocol of behaviour regarding resubmission of rejected requests. The protocol should not have a decisive effect on the architecture's performance (since the Bandwidth Broker can not a priori assume that the clients will follow the instructions laid out by the protocol anyway), and therefore it is kept relatively simple, as can be seen in the pseudocode below. The basic idea is that the client will resubmit the request only if the Bandwidth Broker has indicated that the request should indeed be resubmitted, and in addition if the user is willing to compromise for a possibly delayed reservation.

```
n = 1
while (request is active and request not accepted)
  n = n + 1
  if (BB proposed this request to be resubmitted)
    if (user wants to resubmit request)
      wait for t_resubmit * n time
      resubmit request
    else
      reject request (set to inactive)
  else
    reject request (set to inactive)
End while
```

In order for the Bandwidth Broker to utilize resubmitted requests, it needs to keep a list L of the standby requests. Moreover, it will actively prioritize such requests in expense of newly received requests, and the prioritization will depend on the duration that a specific user has been waiting and resubmitting a request. This is achieved by the adaptation of the main algorithm presented in Fig. 4.

We have to note here that in terms of computational complexity, it might be preferable to have the clients mark whether their requests are a resubmission

or not, using for example a relevant flag. However, putting such trust at the side of a client might allow misbehaved clients to unfairly prioritize their requests by falsely setting the resubmission flag, thereby compromising the scalability of the concept. Furthermore, the fact that the Bandwidth Broker can keep this list sorted means that it can perform the search queries very quickly and minimize the computational overhead.

The addition to the main algorithm is that special care is taken so that the R set contains as many elements from the L list as possible. Furthermore, the elements in L are sorted so that the oldest request is the first element in the list. This means that whenever a request arrives at the Bandwidth Broker module, it is first calculated how much time the respective client has been at the standby state (by keeping the moment that the request initially arrived at the Bandwidth Broker), and is then accordingly placed at the L list.

For a request that has been resubmitted n times, this means that the total time t_w that has elapsed since the initial request will approximately be

$$t_w = \sum_{i=1}^{n} i \cdot t_{\text{resubmit}} + nC_{\text{avg}} \tag{4}$$

where C_{avg} is the average computation time for the admission control module. Therefore

$$t_w = t_{\text{resubmit}} \frac{n(n+1)}{2} + nC_{\text{avg}}$$

```
while (W_q not empty)
  while (next request has not expired)
    forall i where (b_i > B)
      x_i = 0    // reject overbooking requests
    Solve LP maximization for all links:
    max Σ_{i∈R} b_i (t_end - t_start) x_i
    Σ_{i∈R(t)} b_i x_i ≤ B, forall t ∈ (earliest t_start, latest t_end) in R
    x_i ∈ {0,1}, i∈R
    exit loop
  end while
  if ((C-T)>=T)
    R_size = 1
  else if ((C-T)>0)
    R_size = (1-(C-T)/T)* R_size
  else
    R_size = R_size + (W_q-R_size)*a
  if L_size > R_size
    R = R_size first elements of L
  else
    R = L∪(R_size-L_size) elements of R
end while
```

Fig. 4. Adaptive admission control algorithm with special resubmission handling.

which can be simplified to

$$t_w = t_{\text{resubmit}} \frac{n^2}{2} \tag{5}$$

if we assume $C_{\text{avg}} < < t_{\text{resubmit}}$ for a reasonably computationally capable admission control module. This shows that the waiting time is increased quite rapidly as the number of times that the request is rejected increases. This is a necessary feature of the algorithm in order to avoid constant resubmission of rejected requests in situations were the resources that are available to be allocated are significantly below the requested resources. The mechanism for prioritizing the resubmitted requests however balances this effect and allows rejected requests to be accepted at consequent attempts without overcoming the limiting window for which the resources are useful for the end user.

3.3. Other Issues

For many practical applications, specifying the end time of a request is a valid and reasonable assumption. Examples include prearranged videoconferences, content streaming of known duration, online gaming or banking and business applications. There are however also cases when it is not practical or possible for the user to determine the end time of a request. Since we are also considering these open requests, an additional simple calculation has to be performed in order to determine the t_{end} that will be estimated for an open request, as is shown in the algorithm below:

```
set W_q'={non open requests in W_q}
sort W_q' by d_i=t_end-t_start  i∈W_q'
index = p* W_q'_size
forall open requests in W_q
  t_end = t_start + d_index
```

Assuming the duration of open requests will be on average close to the duration of non-open requests (an assumption which might or might not hold depending on the actual environment), parameter p can be for example set to values around or over 98% in order to make sure that reservations for open requests will only be rarely prematurely interrupted, as will be explained below.

A reasonable assumption we can make is that the durations of both the open and non-open requests will follow a normal distribution, based on the large number of requests and their independence regarding durations. Therefore, a value of 98% for parameter p covers all non-open requests within range of twice the standard deviation from the mean of non-open requests (Fig. 5 [37]). According to the normal distribution cumulative distribution function, the probability that an open

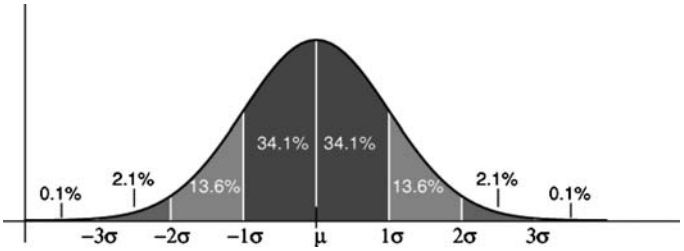

Fig. 5. Normal distribution and standard deviation.

request will exceed its assigned duration is

$$\frac{1}{\sigma_o\sqrt{2\pi}} \int_{-\infty}^{\mu_c+2\sigma_c} e^{\frac{-(u-\mu_o)^2}{2\sigma_o^2}} du \tag{6}$$

where μ_{o}, σ_{o} and μ_{no}, σ_{no} are the mean and standard deviation for open and non-open requests, respectively. It is clear that if the nature of the open requests distribution is close to the non-open one, our mechanism will operate satisfactorily in almost every case. A consideration at this point has to be, that for some environments the final duration of open requests might differ significantly on average from the duration of non-open requests and equation (6) might not produce satisfactory results. Such a case can be avoided by observing the percentage of prematurely interrupted reservations and accordingly raising the cut-off limit for open reservations. This functionality however requires that the request party communicates to the admission module the actual moment when it no longer needs the reservation.

An important issue that is introduced by resubmitted requests, is to make sure that the mechanism for prioritizing resubmitted requests is not misused by non-authorized users. Therefore, the mechanism has to be accompanied by a security architecture that will guarantee the fairness with regard to all valid users in the domain. Fairness in this context is defined as the insurance that submitted requests correspond to actual user needs and comply with the resubmission model described above.

4. EXPERIMENTAL EVALUATION THROUGH SIMULATION

4.1. Evaluation Setup

In order to evaluate the mechanisms proposed in the previous sections, we have used two separate simulation environments. One was a custom simulated system developed for this purpose and the other was based on the popular ns-2

network simulator [38]. The combination of the usage of both these simulated environments allowed us to experiment both in large scenarios and at a high level of detail.

The custom simulated system accepted random requests (requests that did not follow a specific pattern in terms of their arrival time or reservation requests), in order to study the performance of the algorithm and the effect of the computation time threshold and adaptation parameter a on the behavior of the algorithm. Our simulated system was running on an Intel-based PC with 512 MB of memory running Windows 2000. The simulations examined a high-level view of the network, without taking into account details at the packet level since our main focus is on examining the relative performances of the algorithms and isolating these from impact by external or low-level parameters.

The parameters for each request were randomly produced [39] within suitable boundaries (regarding the total duration of each simulation, the total available bandwidth, the minimum and maximum reservation requests) for each situation that we wanted to simulate, and each set of requests designated a specific ingress point at the network (so all requests competed for the same resource limit at the ingress point of the simulated network). In the simulations described in [40], we examined the adaptive properties of the basic algorithm and its behaviour according to the tuning of the configurable parameters.

In order to conduct more realistic simulations at the packet level and obtain more detailed results, we also implemented the algorithms in the popular ns-2 network simulator [38], running on an Intel-based Linux PC with 288 MB of main RAM memory available and a Pentium III Coppermine with 256 KB cache memory on the processor chip, which operated at the frequency of 700 MHz. The parameters for each request were randomly produced within suitable boundaries (regarding the total duration of each simulation, the total available bandwidth, the minimum and maximum reservation requests) for each situation that we wanted to simulate, and each set of requests designated a specific ingress point at the network (so all requests competed for the same resource limit at the ingress point of the simulated network). We simulated a scenario where every request had to specify a steady amount of bandwidth for a specific duration with specific time bounds (there was no possibility for a request to specify a variable bandwidth rate). Randomness was obtained by using the ns-2 RNG class. This class contains an implementation of the combined multiple recursive generator MRG32k3a [41]. The MRG32k3a generator provides 1.8×10^{19} independent streams of random numbers, each of which consists of 2.3×10^{15} substreams. Each substream has a period (i.e., the number of random numbers before overlap) of 7.6×10^{22}. The period of the entire generator is 3.1×10^{57}. More specifically, the random generator was independently generating numbers that were then assigned to each of the attributes for a new request. If the random combination of attributes was invalid (for example the start time of the reservation was after the stop time of

the reservation etc.) the request was discarded as if it had never been generated. Otherwise, it was generated by the node and sent to the Bandwidth Broker for examination. In order to see how each algorithm could scale, we experimented with generating up to 1000 requests in a 50 second interval. Listings of the random requests generated, as well as the source code for replicating our results in ns-2 can be found in [42]. The topology defined at the simulator was a star network, with the Bandwidth Broker module being located in the centre and requests originating from one leaf node towards another leaf node of the network. The links of the simulation represented the provisioning of hoses as defined in [29]. The reason for our choice of the star topology is that once the bandwidth reservation in an actual (not necessarily star) network has taken place according to the hose model, the network can be emulated by a star topology for the purposes of admission control. Any set of feasible connections in the actual network that has made bandwidth reservations according to the hose-model is also feasible in the star topology network, and vice versa.

In order to correctly implement the simulated architecture, it was necessary to make several changes and additions to the ns-2 structure and source code. An agent in ns-2 represents an endpoint where packets are consumed and constructed, using a specific protocol. The Bandwidth Broker that was implemented is based on two new agents, the Edge Bandwidth Broker and the Base Bandwidth Broker. More specifically, the classes BBedgeAgent and BbbaseAgent were derived from class Agent that implements the Edge Bandwidth Broker and the Base Bandwidth Broker. The total bandwidth that the BB manages on each link is determined by the new tcl instruction "set_bndw." A BBedgeAgent, which represents a client (user / application), can send a Resource Allocation Request (RAR) message requesting guaranteed bandwidth between the node where it is running and another node with node-id node_id using the new tcl instruction "sendto." The BBedgeAgent that exists on every node simulates a situation where a BB client is connected to a router on a real network. This agent operates as client that makes the communication with the base BB and updates its local router with the configuration modifications according to new admissions.

4.2. The Effect of Resubmissions

For the first set of evaluations, we examined the percentage of requests that were rejected by the admission control module and how this affected the percentage of requests that the Bandwidth Broker "totally" rejected, meaning that it signaled to the end user that the request should not be resubmitted. In order to get a whole range of values for the rejection rate, we used our custom simulated system to run a large number of simulated experiments with an increasing rate of submissions for the same resources, thus simulating an increase at the rejection rate due to heavy demand for the service. Figure 6 presents the percentage of requests that

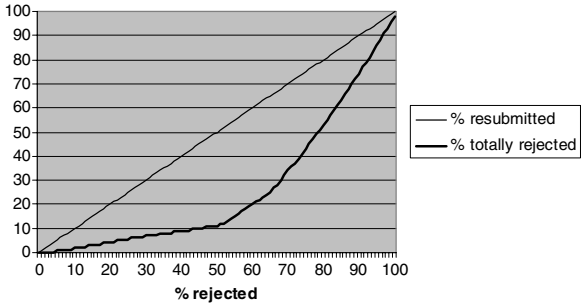

Fig. 6. Percentage of conclusively rejected requests when all rejected requests are resubmitted.

were conclusively rejected when all rejected requests were resubmitted (which means that the end users were constantly trying to get their requests accepted. As can be seen in the figure, the admission control module is tuned in such a way that the Bandwidth Broker signals rejected request for non-resubmission only when the rate of rejections getting higher. Towards the right-hand side of the figure (also when rejections come close to almost all requests submitted), the admission control module chooses to aggressively discourage the end users from resubmitting their requests, in hope that this will enable the request load to lower and hopefully more requests to be accepted in subsequent decisions. Figure 7 presents a slightly different scenario, where we assumed that 50% of the users resubmitted their rejected requests. The Bandwidth Broker again displays similar behaviour, but this time it signals fewer rejected requests to be resubmitted, since resubmissions by the users are fewer. In both scenarios there is a point when about 50% of submissions are rejected, which prompts the admission control module to try more aggressively to discourage resubmissions by end users.

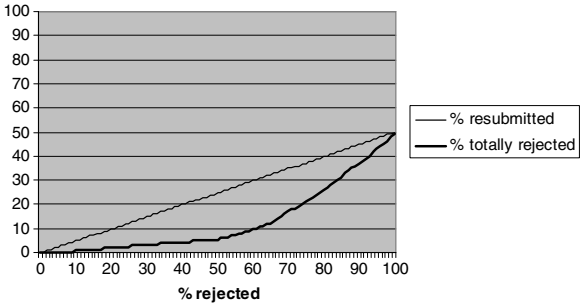

Fig. 7. Percentage of conclusively rejected requests when half of the rejected requests are resubmitted.

Table I. Comparison of Acceptance Rates with/Without Resubmissions

	Number of independent requests	Number of requests submitted to the admission control module	Acceptance rate (% accepted out of total independent requests)
Experiment 1 (without resubmissions)	50	50	30.0%
Experiment 2 (with resubmissions)	50	75	58.0%

For our second set of experiments we intended to compare the acceptance rates and patterns when the resubmission mechanism is active and when it is not. For both experiments in this case we used an input scenario of 50 randomly generated independent requests throughout a time frame of 50 time slots, contesting for available bandwidth of 100 Mbps. Rejected requests were either completely rejected in the first case (no resubmissions), or resubmitted according to the admission control module's suggestion in the second case (resubmissions supported).

The results from the ns-2 simulations (Table I) show a 28% improvement in the acceptance rate with the addition of resubmissions. We have intentionally chosen a scenario where requests are quite densely temporally distributed, in order to make the comparisons between the two cases more valid (if the requests were sparsely distributed the resubmission model would achieve even better acceptance rate). In our scenario therefore, 10 out of 35 originally rejected requests were never resubmitted (because the utilization of the network was already very high and the admission control module decided against resubmitting the rejected requests).

4.3. Comparative Evaluations Using Ns-2

After studying the properties of the adaptive algorithm with and without resubmissions in a more abstract setting and examining the effect of tweaking the adaptation parameter a and the threshold, the next step was to study the operation of the adaptive admission control algorithm (from now on called AAC) and its variation that allows resubmissions (from now on called AACR) at a more realistic setting and compare their performance with other alternatives. In particular, the alternatives we studied are the Simple Admission Control (from now on called SAC), which examines each request on its own and accepts it if there are enough resources to satisfy it, and the Price-Based Admission Control algorithm (from now on called PBAC), which makes a decision on which requests will be accepted trying to optimize the network utilization by gathering and evaluating a group of

requests. In order to solve the NP-complete problem that arises, an approximation algorithm is used which can approximate the optimal solution within a specified range. This type of admission control is similar to the offline version of the algorithm presented in [19]. We have selected the above algorithms for evaluation, because their comparison can provide a good insight in the characteristics we are interested in studying. SAC is the simplest algorithm and by far the most widely used algorithm in related environments. It can therefore be considered as the benchmark algorithm. PBAC lacks adaptive capabilities, allowing us to identify the effect they have on the simulated environment, and finally AAC and AACR differ regarding the support of resubmissions, and therefore their comparative evaluation can reveal the effect of resubmissions on the overall system performance.

The main metrics that we are interested in, in order to compare the performance of the algorithms and evaluate the relative advantages and weaknesses of each, are:

- The acceptance rate, which shows the percentage of requests accepted out of the total number of submitted requests. In case that a flat pricing model is followed (where there is a standard profit per reservation) this metric also corresponds to the network provider's revenue.
- The generated profit for the provider, which is calculated as the product of the bandwidth consumption of each reservation times its duration. This is of course a convention since the pricing model can vary depending on the specific circumstances. We believe though that such a metric is one of the most representative ones, since it can be understood as the amount of resources that is consumed by a reservation and the sum for all reservations shows the network utilization that each algorithm achieves. We are also interested in the average profit achieved per request, which can be in several environments an additional indicator of the effectiveness of the algorithm. In an environment for example when there is an additional overhead to the provider for signaling and allocating a new reservation, it would be beneficial to achieve better network utilization per individual request.
- The delay of being able to deliver either positive or negative answers to the submitted requests.
- The average size of the set of requests examined together, which is a measure of the complexity of the optimization problem solved, and therefore of the overhead to the system.

Our final set of experiments was performed using ns-2 Bandwidth Broker functionality in order to compare the performance of the adaptive algorithm with and without resubmissions throughout a range of request arrival rates. Maximum available bandwidth for the service was set at 100 Mbps, while the duration of each simulation was set at 50 time slots. For algorithms AAC and AACR, the results

Table II. Summary of Results

Averages per algorithm	Acceptance rate	Average delay (time slots)	Average network utilization (transmission rate x time slots)
SAC	29.60%	0	3920014
PBAC	21.79%	7.08	5243307
AAC thr = 5	25.72%	5.44	4532672
AAC thr = 10	24.77%	5.48	4780385
AACR	42.56	5.58	5594577

were obtained setting the adaptation parameter a at a value of 0.2 (moderate adaptation) and a computational threshold of 5 time slots for AACR and both 5 and 10 time slots for AAC.

For each experiment we have measured the percentage of accepted requests, the delay that was required before the Bandwidth Broker would reply to a request and the percentage of network utilization achieved by each algorithm. These results are summarized in Table II.

The following figures display in more detail the behaviour of the algorithms under different experiments with different request frequencies, and they help reveal the features of each algorithm, its relative weaknesses and strengths.

As Fig. 8 demonstrates, the acceptance ratio of all algorithms except AACR remains fairly similar throughout the experiments. SAC is the algorithm that slightly achieves the highest acceptance rate, while PBAC is the one with the lowest, with AAC variations covering the middle. This is not a surprising result, since SAC will always accept a request if there are enough resources available, while PBAC is more oriented towards generating the maximum amount of resource

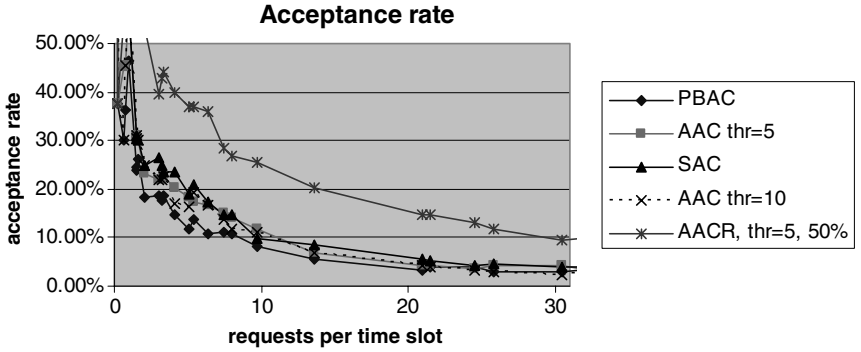

Fig. 8. Acceptance rate.

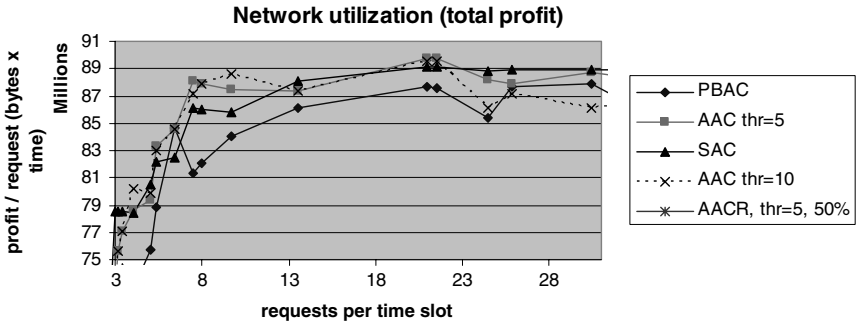

Fig. 9. Network utilization.

utilization, rather than treating all requests alike. Because of the resubmission capability, AACR displays clearly better performance with regard to this metric. This result leads us to the conclusion that in environments where the most significant factor is the satisfaction of the maximum amount of users regardless of their relative weight, the good performance of the SAC algorithm combined with its simplicity make it the most suitable choice. If resubmissions are desirable and can be supported, AACR can then be used for its advantages.

In most cases however, all users will not generate the same revenue for the network provider and a cost scheme will most probably have to take into account both the relative weight of each request, and the effort to maximize the efficiency and utilization of currently available resources. We have tried to cover this aspect with Fig. 9 and Fig. 10, which display the total absolute profit generated for each experiment and the profit per request respectively. We have chosen to measure the provider's profit by calculating the product of a request's duration (in time slots used by the ns-2 simulator) times the resource allocation that a reservation

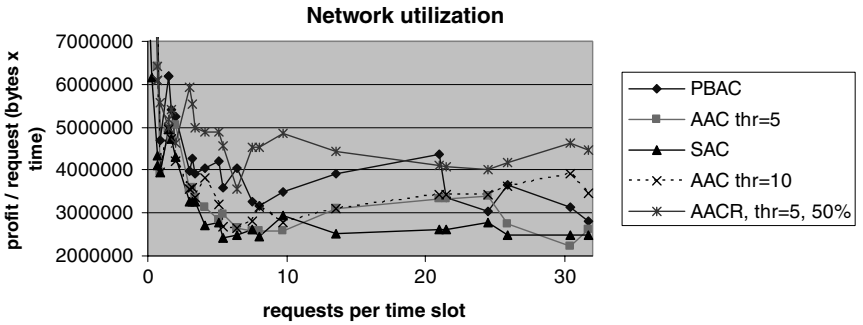

Fig. 10. Network utilization per request.

requires. This metric essentially describes how well the network links are filled with actual traffic, assuming that all simulated users take advantage of the allocated network resources.

We have to mention that in Fig. 9 AACR results are not displayed because they are far larger than all other results, in order to have better distinguishing capability for the rest of the algorithms. These results demonstrate the relative strengths of the price-based approaches, since PBAC is the most efficient algorithm in this regard, followed by AAC, with SAC displaying the worst performance. AAC even surpasses the PBAC performance in several cases when the request arrival ratio increases. The most plausible explanation for this result is that the increased arrival rate of new requests makes the larger size of the set examined by the PBAC algorithm unnecessary. Increasing the threshold for the AAC algorithm seems to have a positive effect on its performance, but comparison with PBAC shows that a restrained increase in the threshold value is enough for obtaining equal or superior results. Therefore, the recommendation for fine-tuning the AAC algorithm is that it is beneficial to increase the threshold value as soon as the arrival rate of request increases. As expected, AACR again displays the best overall performance, which on the case of total profit exceeds several times the results of other algorithms (over twice as much in average).

Figure 10 also implies another characteristic of the algorithms. Starvation of demanding requests by less demanding ones is a valid concern according to the way the adaptive algorithms are structured. However, starvation of demanding requests can be avoided with the resubmission prioritizing mechanism, since a rejected request has better chances of being subsequently considered for admission. Furthermore, Fig. 9 shows that the proposed adaptive algorithms are able to achieve similar or better average size of accepted reservations compared to the

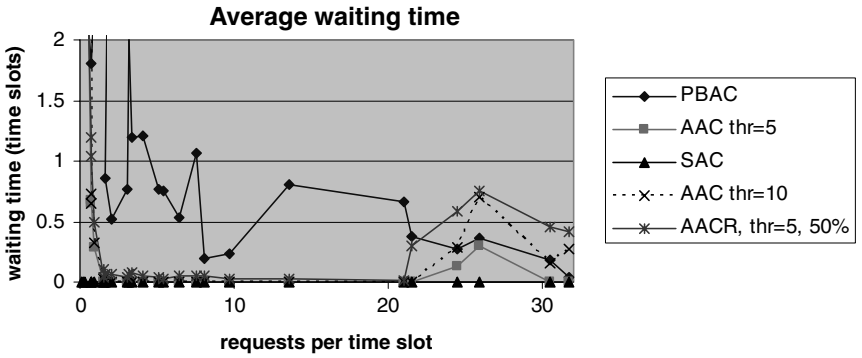

Fig. 11. Average waiting time.

simple benchmarking algorithm SAC, which suggests that larger requests are not in general starved.

In most real environments it is expected that a relatively quick response to a request will be essential. As Fig. 11 demonstrates, SAC is extremely responsive as expected. This also means that there is room for a trade-off that can be used to improve performance in other areas such as the utilization of the network resources. PBAC is not efficient in that regard, as it demands the most time in order to respond to the reservation requests, a situation that in many real-world scenarios is unattainable. The adaptive variations prove to be attractive trade-offs, since for most of the experiments the additional delay they incur is minimal, while at the same time they manage to improve the utilization of the provider's resources, as demonstrated above.

5. CONCLUSIONS

In this paper we have proposed and evaluated an enhanced adaptive admission control algorithm for Bandwidth Brokers. Our proposed algorithm improves on the common admission control modules of Bandwidth Brokers on a number of aspects. It offers better utilization of the network resources, while keeping a balance between simplicity and functionality. It automatically falls back to a simpler model without trying to optimize the network utilization, if its operating environment indicates that the algorithm is too complex for the circumstances. Because resubmitted requests wait for an ever-increasing time period, they do not obstruct the rest of the requests, while each resubmission significantly increases their possibility of being accepted.

Moreover, the admission control module of the Bandwidth Broker tries to prevent unnecessary resubmissions by aggressively discouraging end users from resubmitting requests if it notices that the rejection rate becomes exceedingly high.

A main advantage of our solution is that the admission control module is equipped with the capability of recognizing resubmissions of previously rejected requests and can thus prioritize them. This behaviour results at a smoother variation of the acceptance rate between different types of requests, while the end user also gets a better picture of the domain's load and the success chances of a request.

We believe that the results presented in this paper offer a strong case for the adaptive algorithms (AAC and AACR) in cases where more efficiency in the utilization of the network resources is required, since their adaptive nature incurs minimal overhead and very small delays to the request responses. In that sense, they offer useful alternatives for real world situations by combining the benefits of the simple SAC and the more complicated PBAC algorithm, without introducing any significant drawback of their own.

However, the nature of the studied environments and the multiplicity of factors that affect the outcome of the experiments encourages us to investigate further

into more scenarios that simulate several discrete actual cases and circumstances. Our plans for future work also include the examination and comparative evaluation of the advantages of distributed designs, as well as their impact and overhead for the network.

6. FUTURE WORK

Our future work will focus on further enhancement of the admission control algorithm according to the conclusions from the simulations. An important factor and a point that we intent to further thoroughly investigate regarding resubmissions is the effect that the temporal difference between t_{start} and the moment that the request is submitted has on the overall performance of the admission control module. We also intend to evaluate the basic proposed algorithm and its variations using a real world environment in the framework of a complete implementation of the Bandwidth Broker. Furthermore, we intend to investigate in a real environment the effect of other parameters and practical issues at the operation of a Bandwidth Broker module and the inter-domain Bandwidth Broker operation, such as the necessary level of security, the integration of technological solutions across different administrative and technological domains and the distribution of functionality within a domain and across domains. We also plan to compare the results from actual implementations and from our simulations, in order to evaluate the level of accuracy and realism of the simulator and our extensions to the simulator for the implementations of the algorithms.

REFERENCES

1. RFC 2475 "An Architecture for Differentiated Services," S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, December 1998.
2. RFC 2638 "A Two-bit Differentiated Services Architecture for the Internet," K. Nichols, V. Jackobson, L. Zhang, July 1999.
3. "QBone Bandwidth Broker Architecture," QBone Signaling Design Team, http://qbone.internet2.edu/bb/bboutline2.html (Accessed May 2006).
4. "Bandwidth Broker Implementation," Information and Technology Telecommunication Centre, University of Kansas, http://www.ittc.ukans.edu/∼kdrao/BB/ (Accessed September 2006).
5. T. Braun and G. Stattenberger, *Performance of a Bandwidth Broker for DiffServ Networks, Kommunikation in verteilten Systemen (KiVS03)*, Leipzig, Germany, March 25–28, 2003.
6. J. Ogawa, A. Terzis, S. Tsui, L. Wang, and L. Zhang, *A Prototype Implementation of the Two-Tier Architecture for Differentiated Services*, RTAS99 Vancouver, Canada.
7. L. Wolf, and R. Steinmetz, Concepts for Resource Reservation in Advance, *The Journal of Multimedia Tools and Applications, Special Issue on State of the Art in Multimedia Computing*, pp. 255–278, May 1997.
8. A. Gupta, Advance Reservation in Real-Time Communication Services, In: *Proceedings of the IEEE 22nd Annual Conference on Local Computer Networks* (LCN, 1997), 1997.

9. D. Ferrari, A. Gupta, and G. Ventre, Distributed advance reservation of real-time connections, In: *Proceedings of the Fifth International Workshop on Network and Operating System Support for Digital Audio and Video*, (NOSSDAV'95), Durham, NH, pp. 16–27, April 1995.
10. M. Degermark, T. Kohler, S. Pink, and O. Schelen, Advance Reservations for Predictive Service, In: *Proceedings from 5th Workshop on Network and Operating System Support for Digital Audio and Video* (NOSSDAV'95), Durham, New Hampshire, pp 3–14, April 1995.
11. J. Hwang, S. J. Chapin, H. A. Mantar, I. T. Okumus, Rajeesh Revuru, and Aseem Salama, An Implementation Study of a Dynamic Inter-Domain Bandwidth Management Platform in Diffserv Networks, In: *Proceedings of the 9th IEEE/IFIP Network Operations and Management Symposium* (NOMS 2004), Seoul, Korea, April 2004.
12. K. Wong and K. L. E. Law, ABB: Active Bandwidth Broker In: *Proc. SPIE ITCom* 2001, Vol. 4523, pp. 253–262, Denver, August 2001.
13. N. Blefari-Melazzi, J. N. Daigle, and N. Femminella, *Stateless Admission Control for QoS Provisioning for VoIP in a DiffServ Domain, 18th International Teletraffic Congress*, Berlin, Germany, September 2003.
14. M. Menth, S. Gehrsitz, and J. Milbrandt, *Fair Assignement of Efficient Network Admission Control Budgets, 18th International Teletraffic Congress*, Berlin, Germany, September 2003.
15. C. Brandauer, W. Burakowski, M. Dabrowski, B. Koch, and H. Tarasiuk, *AC algorithms in Aquila QoS IP network, 2nd Polish-German Teletraffic Symposium PGTS 2002*, Gdansk, Poland, September 2002.
16. M. Dabrowski, A. Beben, and W. Burakowski, *On Inter-Domain Admission Control Supported by Measurements in Multi-domain IP QoS Network, Inter-Domain Performance and Simulation* IPS 2004, Budapest, Hungary, March 2004.
17. S. Machiraju, M. Seshadri, and I. Stoica, A Scalable and Robust Solution for Bandwidth Allocation, Technical Report UCB//CSD02 -1176, University of California at Berkeley, 2002.
18. A. Greenberg, R. Srikant, and W. Whitt, Resource Sharing for Book-Ahead and Instantaneous-Request Calls, *IEEE/ACM Transactions on Networking*, vol 7, No. 1, pp. 10–22, February 1999.
19. C. Chhabra, T. Erlebach, B. Stiller, and D. Vukadinovic, Price-based Call Admission Control in a Single DiffServ Domain, TIK-Report Nr. 135, May 2002.
20. C. Bouras, V. Kapoulas, G. Pantziou, and P. Spirakis, Competitive Video On Demand Schedulers for Popular Movies, *Discrete Applied Mathematics*, Vol. 129, pp. 49–61, 2003.
21. E. Mykoniati, C. Charalampous, P. Georgatsos, T. Damilatis, D. Goderis, P. Trimintzios, G. Pavlou, and D. Griffin, Admission Control for Providing QoS in DiffServ IP Networks: The TEQUILA Approach, *IEEE Communications Magazine*, Vol. 41 No. 1, 2003.
22. M. Menth, Efficient Admission Control and Routing for Resilient Communication Networks, PhD thesis, University of Würzburg, Faculty of Computer Science, Am Hubland, July 2004 http://opus.bibliothek.uni-wuerzburg.de/opus/volltexte/2004/994/pdf/Menth04.pdf (Accessed September 2006).
23. R. Boutaba, W. Szeto, and Y. Iraqi, DORA: Efficient Routing for MPLS Traffic Engineering, *Journal of Network and Systems Management, Special Issue on Internet Traffic Engineering and Management*, Vol. 10 No. 3, pp. 309–325, 2002.
24. European Telecommunications Standards Institute (ETSI), http://www.etsi.org/ (Accessed September 2006).
25. ETSI ES 282 003 V1.1.1. Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); Resource and Admission Control Sub-system (RACS); Functional Architecture, European Telecommunications Standards Institute, 2006.
26. C. Gallon, Quality of Service For Next Generation VoIP Networks, MSF-TR-QoS-FINAL.
27. C. Gallon, and Olov Schelén, Bandwidth Management in Next Generation Packet Networks, MSF Technical Report, MSF-TR-ARCH-005-FINAL, August 2005.

28. 3GPP TS 23.207 v6.4.0 3rd Generation Partnership Project; Technical Specification Group Services And System Aspects; End-to-end Quality of Service (QoS) concept and architecture (Release 6).
29. N. Duffield, P. Goyal, A. Greenberg, P. P. Mishra, K. K. Ramakrishnan, and J. E. van der Merive, Flexible Model for Resource Management in Virtual Private Networks, SIGCOMM, pp. 95–108, 1999.
30. R. J. Gibbens and P. J. Hunt, Effective Bandwidths for the Multi-Type UAS Channel, *Queueing Systems*, Vol. 9, pp. 17–28, 1991.
31. R. Guerin, H. Ahmadi, and M. Naghshineh, Equivalent capacity and its application to bandwidth allocation in High-Speed Networks, *IEEE J. Select. Areas Commun*, Vol. 9, pp. 968–981, 1991.
32. F.P. Kelly, Effective Bandwidths at Multi-Class Queues, *Queueing Systems*, Vol. 9, pp. 5–16, 1991.
33. V. Vazirani, *Approximation Algorithms*, Springer-Verlag, 2001.
34. B. Lee, W.-K. Woo, C.-K. Yeo, T.-M. Lim, B.-H. Lim, Y. He, and J. Song, Secure Communications between Bandwidth Brokers, *Operating Systems Review*, Vol. 38, no. 1, pp. 43–57, 2004.
35. V. Sander, The security Environment of SIBBS, http://qbone.internet2.edu/bb/SIBBS-SEC.doc, June 2000.
36. V. Sander, W. Adamson, I. Foster, and A. Roy, End-to-End Provision of Policy Information for Network QoS, Proceedings of the 10th International Symposium on High Performance Distributed Computing, pp. 115–126, 2001.
37. Pat Beirne illustration, 5/1/06, based on an original graph created by Jeremy Kemp, 2/9/05, http://en.wikipedia.org/wiki/Image:Standard_deviation_diagram.png (Accessed September 2006).
38. The Network Simulator – ns-2, http://www.isi.edu/nsnam/ns/ (Accessed September 2006).
39. www.random.org.
40. C. Bouras, and K. Stamos, *An Adaptive Admission Control Algorithm for Bandwidth Brokers, 3rd IEEE International Symposium on Network Computing and Applications (NCA04)*, Cambridge, MA, USA, pp. 243–250, August 30 – September 1 2004.
41. P. L'Ecuyer, Good Parameters and Implementations for Combined Multiple Recursive Random Number Generators, *Operations Research*, Vol. 47, no. 1, 159–164, 1999.
42 . http://ru6.cti.gr/ru6/ns_home.php (Accessed September 2006).

Christos Bouras obtained his Diploma and PhD from the Computer Science and Engineering Department of Patras University (Greece), where he is currently an Associate Professor. He is also a scientific advisor of Research Unit 6 in Research Academic Computer Technology Institute (CTI), Patras, Greece.

Kostas Stamos obtained his Diploma and Master Degree from the Computer Engineering and Informatics Department of Patras University. He is currently a PhD candidate at the same department and is working at Research Unit 6 in Research Academic Computer Technology Institute (CTI), Patras, Greece.