Resubmissions and Partly Defined Requests in an Adaptive Admission Control Algorithm for Bandwidth Brokers

Ch. Bouras K. Stamos
Research Academic Computer Technology Institute, PO Box 1122, Patras, Greece and Computer Engineering and Informatics Dept., Univ. of Patras, GR-26500 Patras, Greece
Tel:+30-2610-{960375, 990316}
Fax:+30-2610-{969016, 960358}
e-mail: {bouras, stamos}@cti.gr

Abstract

The purpose of this paper is to examine the issues related to the efficiency and adaptability of the admission control module of a Bandwidth Broker. The proposed architecture is able to handle situations such as the resubmission of previously rejected requests and offer a fair treatment for requests in an environment where a subset of these requests do not specify their ending time. We describe the proposed mechanisms for such cases which aim at solving the additional problems of fairly prioritizing resubmitted requests and efficiently handle requests which do not specify ending times. We also provide experimental evaluations of the proposed algorithms and the conclusions they lead us to.

1. Introduction

Bandwidth Brokers are entities proposed in the framework of the DiffServ architecture for Quality of Service (QoS) provision in the Internet. The Bandwidth Broker [1] manages the resources within the specific domain by controlling the network load and by accepting or rejecting bandwidth requests. Bandwidth Brokers are an intensely studied field, and a number of architectures have been proposed for the various aspects of its operation ([2], [3], [4], [5]). Especially 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 [6], [7] and [8]. Reference [9] proposes a novel architecture for the admission control module that aims at maximizing the resource utilization for the network provider, while keeping the computation requirements of the admission control module of the Bandwidth Broker relatively low. Initial experimentation based on simulations has also been provided, dealing with the adaptive capabilities of the proposed architecture. Reference [10] presents an inter-domain approach to the admission control.

In this paper, we examine an adaptive admission control module and the improvements in the architecture's efficiency that it can lead to. We also examine how the admission control module could handle or take advantage of 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.

The rest of the paper is organized as follows: Section 2 discusses the adaptive admission control algorithm. An enhanced algorithm is described and analyzed in section 3 and section 4 discusses further related issues. Our performance evaluations are presented in section 5, and section 6 summarizes our conclusions and our future work in this area.

2. Adaptive admission control

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 Figure 1.

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 [11] 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 and that part of the available bandwidth for the links has been assigned to the management of the Bandwidth Broker for the DiffServ service.

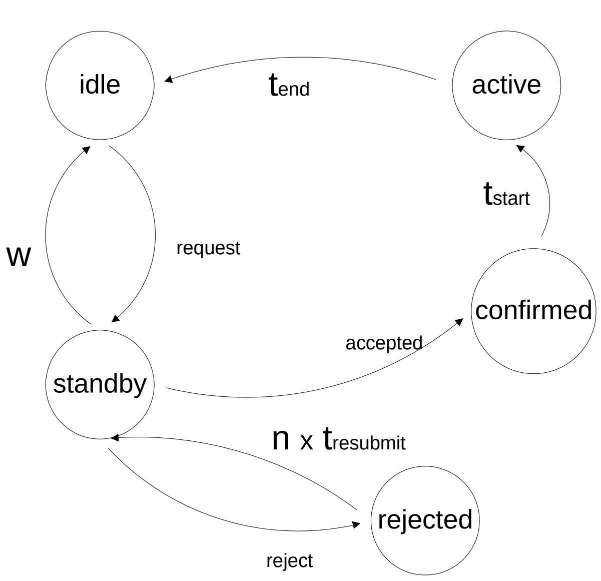

Figure 1. Request states

Our approach also decouples the admission control decision from the routing issues. This means that core routers are not involved in the process of admission control and signaling. 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 for reservation of resources within the domain managed by the Bandwidth Broker has the form of

$r(t_{start}, t_{end}, b, w, in, out)$

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. The w parameter gives the user submitting the request additional flexibility, because it can determine what is the latest time that the user expects a response. Since the rejection of a request can have a significant impact on a user's activities, we have to avoid delaying the notification in order to gather more requests and increase the admission module's effectiveness. Parameter w plays a central role in the implementation of our algorithm, because it gives the admission control module the possibility to take advantage of simultaneous examination of multiple requests, without risking excessive consequences on the user's schedule. 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 (the list is easily maintained sorted by inserting each new request according to its w parameter). *in* and *out* are the ingress and egress nodes where the request flow enters and leaves the domain. Whether the request is inter-domain or intra-domain is transparent at this stage. Our model does not presume a specific type of inter-Bandwidth Broker communication, and intra-domain requests are more important in determining the admission module's performance characteristics since they are not affected by the inter-domain coordination protocols for the Bandwidth Brokers (how to route inter-domain requests, how to handle their rejections, etc.).

As soon as the first item, say r_1 (with the closest w to the current time) in W_q 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

$$R=\{r_1, r_2, \ldots, r_m\} \subseteq W_q$$

and we want to find a subset $R_c \subseteq R$ such that $\Sigma_{ri \in R} cb_i \leq B$ at any time point where B is the total available bandwidth for the service and try to maximize $\Sigma_{ri \in R} cb_i$ 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 [12] and therefore it is proposed to use an approximation algorithm to solve the following linear programming relaxation in polynomial time and then make the solution discrete regarding the variables x_i, which represent whether r_i is accepted or not.

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. 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.

Below is a short summary of the main algorithm in pseudocode:

```
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 $\Sigma_{i \in R} b_i(t_{end}-t_{start}) x_i$
    $\Sigma_{i \in R(t)} b_i x_i \leq B$, forall t ∈ (earliest $t_{start}$,
                               latest $t_{end}$) in R
    $x_i \in \{0,1\}$, $i \in R$
    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
```

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 variables *x_i*, represent whether *r_i* is accepted or not.

This algorithm 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) leads the algorithm to quickly converge to the best approximation of the optimal solution. In particular, solving the recursive function for *R_{size}* proves that it 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 [9].

3. Algorithm enhancements

It is possible to enhance the algorithm with examining resubmissions of previously rejected requests. There is no change needed 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 req 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 below.

```
while ($W_q$ not empty)
  while (next request has not expired)
    forall i where ($b_i$ > B)
      $x_i$ = 0   // reject overbooking reqs
    Solve LP maximization for all links:
    max $\Sigma_{i \in R} b_i(t_{end}-t_{start}) x_i$
    $\Sigma_{i \in R(t)} b_i x_i \leq B$, forall t ∈ (earliest
                    $t_{start}$, latest $t_{end}$) in R
    $x_i \in \{0,1\}$, $i \in R$
    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 = $L \cup (R_{size}-L_{size})$ elements of R
end while
```

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.

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

$$t_w = \sum_{i=1}^{n} i \cdot t_{resubmit} + nC_{avg} \qquad (1)$$

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

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

if we assume $C_{avg} << t_{resubmit}$ 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.

4. Discussion – 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 $d_i=t_{end}-t_{start}$ $i \in W_q'$
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 95% in order to make sure that reservations for open requests will only be rarely prematurely interrupted.

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. In order to avoid a higher than expected ratio of prematurely interrupted reservations for open requests, the Bandwidth Broker's admission control mechanism keeps track of the prematurely interrupted reservations. If the interruption ratio is indeed higher than expected, the Bandwidth Broker module can simply set W_q' to be the set of open requests that have already concluded their reservation without premature interruption (so that their final duration is known). 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.

5. Performance

In this section, we focus on the improvements in efficiency and clients' request satisfaction that are gained by applying the mechanisms described in this paper. More specifically, we want to study the relative strengths and weaknesses of the adaptive mechanism and examine what is the effect of allowing (and actively prioritizing) the resubmission of previously rejected requests.

In order to conduct realistic simulations at the packet level and obtain detailed results, we implemented the algorithms in the popular ns-2 network simulator [13] and run a set of ns-2 simulated experiments 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 700MHz. 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 [14]. 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. Listings of the random requests generated, as well as the source code for replicating our results in ns-2 can be found in [15]. The used topology for all experiments was a simple 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.

Our first set of experiments was designed in order 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).

Table 1. Comparison of acceptance rates with/without resubmissions

	Indepen-dent requests	**Requests submitted to the adm. control module**	**Acceptance rate (% accepted out of total independent requests)**
Exp 1 (no resubmissions)	50	50	30.0%
Exp 2 (with resubmissions)	50	75	58.0%

The results from the ns-2 simulations 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).

Our next set of experiments was performed in order to compare the performance of the adaptive algorithm with and without resubmissions throughout a range of request arrival rates. Figure 2 displays the network utilization (measured in bytes x time reserved by requests) for the two adaptive variations presented in this paper, and compares them with a simple admission model where a request is simply accepted if there are available resources for it at the time of the request. The results show that the capability of resubmissions can be very effective in significantly increasing the effectiveness of the admission procedure and the utilization of the network resources. If the operating environment does not allow for resubmissions, Figure 2 demonstrates that our adaptive approach still bears significant advantages (an increase of almost 40% in most cases) in the network utilization metric.

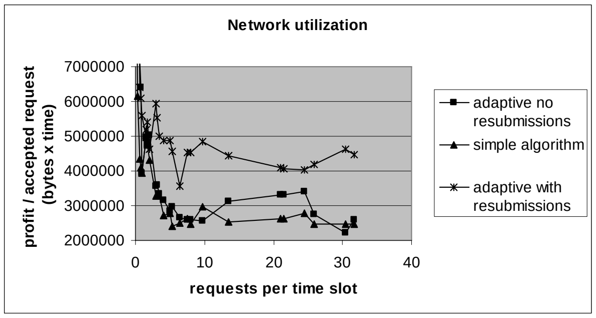

Figure 2. Network utilization

One of the biggest concerns for adopting an adaptive price-based approach to the admission control issue is the increased time that will elapse before a request receives an answer. Figure 3 deals with this aspect and demonstrates that our ns-2 experiments indicated a very low increase at the average waiting time for most intermediate cases. Only when the request arrival rate is very low or very high is the increase significant. In the former case the reason is that the admission control module "buffers" received requests waiting for additional requests in order to be evaluated together, and because requests arrive sparsely, average waiting time gets higher. In the latter case the reason seems to be the larger instances of the optimization problems that have to be solved by the admission control modules. Here, the adaptive capability of the proposed approaches helps reduce this effect.

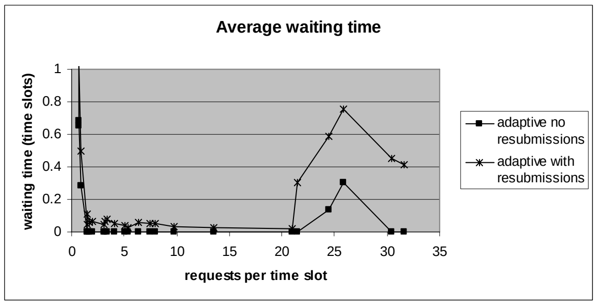

Figure 3. Average waiting times

6. Conclusions - Future work

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.

Our future work will focus on further enhancement of the admission control algorithm according to the conclusions from the simulations. We intend to evaluate the algorithm using a real world environment in the framework of a complete implementation of the Bandwidth Broker. 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.

7. References

[1] RFC 2638 "A Two-bit Differentiated Services Architecture for the Internet", K. Nichols, V. Jackobson, L. Zhang, July 1999

[2] "QBone Bandwidth Broker Architecture", QBone Signaling Design Team, http://qbone.internet2.edu/bb/bboutline2.html

[3] "Bandwidth Broker Implementation", Information and Technology Telecommunication Centre, University of Kansas, http://www.ittc.ukans.edu/~kdrao/BB/

[4] T. Braun, G. Stattenberger, "Performance of a Bandwidth Broker for DiffServ Networks", Kommunikation in verteilten Systemen (KiVS03), Leipzig, Germany, March 25-28, 2003

[5] J. Ogawa, A. Terzis, S. Tsui, L. Wang, L. Zhang. "A Prototype Implementation of the Two-Tier Architecture for Differentiated Services", RTAS99 Vancouver, Canada

[6] N. Blefari-Melazzi, J. N. Daigle, N. Femminella, "Stateless Admission Control for QoS Provisioning for VoIP in a DiffServ Domain", 18th International Teletraffic Congress, Berlin, Germany, September 2003

[7] M. Menth, S. Gehrsitz, J. Milbrandt, "Fair Assignement of Efficient Network Admission Control Budgets", 18th International Teletraffic Congress, Berlin, Germany, September 2003

[8] C. Brandauer, W. Burakowski, M. Dabrowski, B, Koch, H. Tarasiuk, "AC algorithms in Aquila QoS IP network", 2nd Polish-German Teletraffic Symposium PGTS 2002, Gdansk, Poland, September 2002

[9] C. Bouras, K. Stamos, "An Adaptive Admission Control Algorithm for Bandwidth Brokers", 3rd IEEE International Symposium on Network Computing and Applications (NCA04), Cambridge, MA, USA, August 30 - September 1 2004, pp. 243-250

[10] M. Dabrowski, A. Beben, 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

[11] N. Duffield and P. Goyal and A. Greenberg and P. P. Mishra and K. K. Ramakrishnan and J. E. van der Merive, "Flexible Model for Resource Management in Virtual Private Networks", SIGCOMM 1999, pp. 95-108

[12] V. Vazirani, "Approximation Algorithms", Springer-Verlag, 2001

[13] The Network Simulator - ns-2, http://www.isi.edu/nsnam/ns/

[14] Pierre L'Ecuyer. Good parameters and implementations for combined multiple recursive random number generators. Operations Research, 47(1):159–164, 1999.

[15] http://ouranos.ceid.upatras.gr/diffserv-ns/intro.htm