Performance Evaluation Of Data Structures For Ad-mission Control In Bandwidth Brokers

Lars-O. Burchard, Hans-U. Heiss
Technical University of Berlin
Franklinstrasse 28/29, 10587 Berlin, GERMANY
{baron,heiss}@cs.tu-berlin.de

Keywords: Bandwidth Broker, Advance Reservation, Data Structures, Admission Control, Quality-of-Service

Abstract

Bandwidth brokers act as a management component in networks and allow to enable resource reservations (QoS guarantees). In a bandwidth broker, fast and efficient admission control is an important task which might completely dominating the broker's response time to a reservation request. In case the reservation is made in advance, i.e. a longer period of time before the actual requirement for QoS is established (advance reservation), it is necessary to use data structures allowing to cover longer time periods. In this paper, an array and a segment tree specially designed for the admission control task were examined. The tree was extended in order to achieve a better memory utilization and shorter admission times compared to the original implementation. Nevertheless, the evaluation shows the superiority of arrays concerning both memory requirement and admission time for nearly any of the examined scenario.

1 INTRODUCTION

Bandwidth brokers are used as a managing system in differentiated services (Diff-Serv) networks. A broker processes reservation requests and grants or denies access to the network. A single broker is only responsible for managing a certain part of a network (domain), e.g. a corporate network or the network of an ISP.

Two possible types of bandwidth reservations can be distinguished: immediate and advance reservations. In contrast to immediate reservations, where the network reservations are established immediately after the request is admitted, advance reservations allow to specify and request a given quality-of-service (QoS) for a transmission a long time before the actual transmission has to be made. In the following, the time for which reservation requests can be made is called book-ahead time.

One important task to be performed by a bandwidth broker in such an environment is admission control, which means reservation requests have to be checked whether sufficient resources are available for the duration of the reservation. In the scenario examined in this paper, only advance reservations were considered. Admission control for immediate reservations is less complex, in particular it is just a special case without book-ahead time.

In order to efficiently perform the admission control task, it is necessary to find out which data structures are most suitable for establishing fast admission control, i.e. to minimize the response time of
a broker for a single reservation request. In addition to that, obviously the memory requirement of the data structures influences the performance of the bandwidth broker and therefore has to be taken into account. Another aspect of the considerations has to be how the chosen data structures can deal with the fact that the book-ahead time changes dynamically during run-time. This trivial fact is rather important in the environment considered and it will be shown in the following sections that it influences both the admission speed and especially memory requirement.

In this paper, two data structures (a specially designed tree and an array) were examined in a number of different scenarios with respect to the issues previously described, i.e. admission time, memory requirement, and suitability for dynamically changing book-ahead time. We extended the original tree implementation in order to achieve a more efficient memory usage. Our evaluations show, that arrays perform far better in the examined scenarios. Concerning both response time and memory usage, arrays are superior to the tree structure and moreover, an array can be more easily adapted to deal with the changing book-ahead time without loss of speed or memory efficiency.

In [2], which is a more detailed version of this paper, additional test results and information about the data structures can be found.

2 RELATED WORK

Quality-of-service in networks has been in the focus of researches for a variety of reasons. For some applications such as audio/video streaming it is required in order to avoid network jitter and congestion. Moreover, when it is necessary to deliver a large amount of data over a network within a given time, it is required to reserve network resources in order to meet the given deadline. This is the case in grid computing environments and also distributed multimedia applications [3].

The requirement for advance reservations has been identified in several papers [4, 5]. Those early works mainly concentrate on design issues for enabling advance reservations and present extensions to existing network reservation protocols such as RSVP [1].

Most recent works deal with using differentiated services and introduce the concept of bandwidth brokers for managing network resources [6]. In [7] the basic principle of wide scale deployment of DiffServ using two service classes (premium and best-effort service) on the Internet using bandwidth brokers is described. This is the approach we chose for the bandwidth broker presented in this paper. The underlying network is supposed to support both differentiated services and MPLS [9]. MPLS allows using constraint based routing and therefore is ideally suited to support traffic engineering using a bandwidth broker.

The considerations presented in this paper are based on [8]. The segment tree described there is the foundation for our own implementations and tests. It was extended and developed further concerning both admission speed and especially memory consumption. However, our tests show arrays are better suited in the environments considered.

3 DATA STRUCTURES

In the following sections, the data structures and their properties are described, starting with a brief overview of the admission control process.

3.1 Admission Control

The bandwidth broker which is described in this paper supports advance reservations which means, there is a book-ahead time for which reservations can be issued (see Figure 1). A reservation request is de-
defined by its starting and finishing time and the peak bandwidth requirement. The admission control procedure is to determine whether sufficient bandwidth is available for the reservation period.

Figure 1: Advance Reservations and Book-ahead Time

The data structures examined in this paper are implemented using slotted time. This means, the starting and the finishing time have to be defined using a certain granularity, which cannot be changed during run-time. For the tests presented in this paper, a granularity of one minute was chosen.

3.2 Array

Arrays are easy to implement and have a low complexity thus allowing extremely fast access to each array element. In our array implementation, one element of the array represents each time slot. Each element stores the accumulated bandwidth allocated for the respective slot. This allows to quickly access each time slot and to determine whether sufficient bandwidth is available. In order to perform admission control using the array, all the time slots within the starting and finishing time given by the reservation request have to be checked (see Figure 2). One important property of arrays is, that the memory requirement is constant and does not change over time. Assuming 4 bytes per slot, and covering n slots arrays require a total of \( n \times 4 \) bytes.

3.3 Segment Tree

The segment tree was originally described in [8]. Due to the space limitation, we do not discuss details here. For the description please refer to the original document.

One of the major drawbacks of the segment tree is the memory consumption. Depending on the actual implementation, segment trees require up to 10 times more memory than arrays (see Section 4.5). Including pointers to child nodes and assuming 4 bytes per pointer and 4 bytes per integer, each tree node requires 20 bytes. Thus, the worst-case memory requirement for a tree is covering \( n \) slots \( (n \times 2 - 1) \times 20 \) bytes.

The original tree implementation was extended using from the observation that only a limited number of tree nodes is actually required even in case many reservations are stored within the tree. The memory requirement during run-time is reduced by only allocating the nodes that are actually required. At the beginning only the root node is present. In case a new reservation is added, missing tree nodes are generated on-demand. However, this does not affect the worst-case memory requirement.

3.4 Dynamic Book-Ahead

Another drawback of the segment tree is its unsuitability for dynamically advancing book-ahead time. In order to cope

---

\(^1\)This seems to be sufficient for most types of reservations. However, if required the value can be changed according to the respective requirements.
4 EVALUATION

In this section, the results of our tests are presented. Admission speed and memory requirement are examined in different scenarios.

The experiments were made for the single data structures and a single link as well as for a multi-link scenario which allows to study the impact of the performance on the complete admission control process of a bandwidth broker.

4.1 Performance of the Data Structures

At first, the admission time and memory consumption of the data structures were examined\(^2\). Each reservation request was made on an empty data structure, i.e. no other reservations were present. The results can be seen in Figure 5.

---

\(^2\)As a function of the duration of the reservation request
It can be observed that regarding admission speed, arrays are always competitive (compared to the segment tree) up to a reservation length of approximately 400 slots although the admission time of arrays grows proportionally with the duration of the request.

It is somewhat surprising that the tree traversal is relatively time intensive even for short reservations. During the admission process, the tree implementation requires the traversal of several levels of the tree whereas arrays have the advantage of constant access to the single slots. The number of levels and therefore nodes visited during the admission decision is relatively high for short reservations thus requiring significant computing time.

With respect to memory usage, the data structures are relatively unaffected by the duration of the requested reservation. Storing only a single reservation request in the case arrays obviously require far more memory than the tree implementation which dynamically allocates the required memory. However, as we will show in the following sections this changes drastically in case a large number of reservations is stored.

Arrays have advantages when the duration of a particular reservation request is rather short whereas trees are less sensitive to the duration of the requested reservation. In the following sections, it will be shown that arrays are superior even when the average of the requested durations is larger than 400 slots.

4.2 Link Admission Control

The admission control process for links includes two phases: a check phase (testing whether sufficient bandwidth is available) and in case of success an update phase (adding the requested bandwidth to the respective data structure).

The measurements were performed using two types of reservation requests according to two different sets of parameters. The first parameter set is the same as used in [8]: a book-ahead time of 30 days, slotted time using a granularity of five minutes, i.e. a total of 8640 slots, and reservation durations which are uniformly distributed in [20..180] minutes (i.e. 4 - 32 slots). In order test a wider scenario a second set of parameters was used: a book-ahead time of 65536 minutes (i.e. approximately 1.5 months), a granularity of one minute, and reservation durations which are uniformly distributed in [10..2000] minutes.

In the following sections, each point in the figures represents the average of 1000 consecutive admission decisions.

4.3 Single-Link Admission Control

In this section, the admission time and memory consumption of the data structures for a single link are shown.

![Figure 6: First Parameter Set: Admission Speed (above) and Memory Usage (below)](image)

The results are shown in Figure 6 for the first parameter set and in Figure 7 for the second parameter set. It can be
observed, that the admission speed of arrays is higher in both cases which confirms the results presented in Section 4.1. Only, when the number of rejected reservations increases the difference between array and segment tree closes substantially because only the first phase (check) is required. However, arrays are still faster. The memory requirement of trees compared to that of arrays is more than seven times higher. When using static allocation of the tree nodes, the gap between the memory consumption of trees and arrays is even wider.

![Comparison of array and segment tree admission speed](image1)

![Comparison of array and segment tree memory usage](image2)

Figure 7: Second Parameter Set: Admission Speed (above) and Memory Usage (below)

It can be observed that compared to the tests using first set of parameters, the difference between the admission speed using arrays and trees respectively decreases for successful reservations. This is caused by the increased length of the requested reservations. However, the array implementation still requires significantly less admission time than the segment trees, i.e., arrays also benefit from rejected requests.

The memory consumption of the segment tree is again much higher than that of the array and the gap between both data structures widens.

### 4.4 Multi-Link Admission Control

In this section, we present and discuss the results of the performance measurements for the complete admission control process in a network with multiple routers. We used a network of 9 routers and 14 links. For each outgoing interface, admission control has to be performed, i.e., for a total of 28 interfaces. Since the segment trees performed worse under the first parameter set, only the second parameter set was used in these tests.

![Multi-Link Admission Speed](image3)

![Multi-Link Memory Usage](image4)

Figure 8: Multi-Link: Admission Speed (above) and Memory Usage (below)

The admission times and total memory consumption of all links are shown in Figure 8. In general, the results confirm the results presented in the previous section: arrays have a significant advantage over segment trees. When the number of rejections rises, the gap between the admission speed of the array and the segment tree closes. However, the decision
times of segment trees are nearly always above those of the array.

Using our segment tree implementation, links with less utilization do not require a considerable amount of memory. However, our simulation shows, that even when some links are less utilized, the rest of the links (with higher utilization) contribute to the total memory consumption exceeding that of arrays. It can be seen that the peak memory allocation of trees after 1 million requests exceeds the constant requirement of arrays by a factor of up to 10.

4.5 Dynamic Book-ahead

In the previous sections, reservation requests occurred within a single data structure, i.e. for a static time pointer of 0. However, in a real-world scenario during the run-time of the bandwidth broker the book-ahead time advances and the data structures must be able to deal with that in an appropriate way (see Sec. 3.4). In this section, both data structures are examined in a scenario with dynamic book-ahead time using multiple links.

The effects of the advancing time on admission speed and memory usage can be seen in Figure 9. The results in this scenario were generated starting with a slot time of 0 (i.e. at the "beginning" of the first tree) and advancing the book-ahead time by one slot after each 10 admission requests. When using trees, consequently at the beginning only the first tree fills up. After a while also the second tree stores reservations, and finally reservation requests only affect the second tree.

 Compared to the admission speed in the static case, considerable changes cannot be observed, although in many cases a reservation request had to be divided in two part thus requiring twice the admission time.

Figure 9 also shows the effect of dynamic book-ahead time on the total memory requirement of the trees: compared to the memory usage of just a single tree about 30% additional memory is required, i.e. about 13 times more than required by the arrays. Although the first tree is completely deleted during run-time this does not significantly reduce the overall memory requirement.

In general, it can be said that the disadvantages of the segment tree increase in the dynamic scenario.

5 CONCLUSION

In this paper, we presented data structures and performance tests of the admission control process in a bandwidth broker. Two data structures, array and a segment tree, were evaluated. The original implementation of the segment tree was further improved in order to reduce memory requirement and increase the decision speed.

The observed behavior of the data structures leads to the surprising conclusion, that arrays are better suited for the admission control task in many scenarios.
The experiments described in this paper show that arrays are inferior when the average duration of admitted reservation requests is rather long (i.e. significantly above 1000 slots) and only few reservations are rejected.

In any case where the available network resources are highly utilized, i.e. the probability for a request being rejected is high, even reservations with duration above 1000 slots can be handled efficiently using arrays which in these cases require only a short decision time.

Even in case the admission speed using the segment tree will be further increased, the memory requirement of arrays will always remain significantly below that of segment trees, especially in the dynamic scenario where even two trees are required. In addition to that, the dynamically changing book-ahead time can be implemented easier using arrays.

References


