Performance Evaluation of Data Structures for
Admission Control in Bandwidth Brokers

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

Abstract
Bandwidth brokers allow to enable resource reservations (QoS guarantees) in networks. In a bandwidth broker, fast and efficient admission control is an important task which almost completely dominates the broker’s response time to a reservation request. In case the reservation is to be 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 (which was specially designed for the admission control task) were examined. The segment 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 almost any length and distribution of reservations.
1 Introduction

In order to enforce quality-of-service (QoS) guarantees on the network, two types of technologies are available. In contrast to the integrated services approach (IntServ) which lacks scalability, the differentiated services (DiffServ) model allows to enable QoS on a wide scale such as the Internet. However, using DiffServ requires an additional management component in order to facilitate end-to-end QoS guarantees. Bandwidth brokers are considered as a managing system in such networks: a client (person or software) sends a reservation request to the broker, which then 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.

In general, two types of bandwidth reservations are possible: 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 QoS for a reservation a long time before the actual transmission is 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 in order to establish 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 also 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 at 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 admission speed and especially memory requirement.

In this paper, two data structures (a specially designed tree and an array) were examined in several different scenarios with respect to the issues previously described, i.e. admission time, memory consumption, 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. For example, the experiments showed that the memory requirement of the tree was approximately 10 times higher compared to an array. Moreover, in contrast to the tree an array can be easily adapted to deal with the changing book-ahead time without loss of speed or memory efficiency.

The data structures were implemented in a prototype bandwidth broker, running in a laboratory of the Technical University of Berlin.

In the following sections, after discussing some related work, the details of the admission control process are described. Following that, both data structures are presented together with their basic properties and how the structures are applied in the bandwidth broker. In section 6 the performance of both data structures is compared for single-link and multi-link admission control. In section 6.4, it will be shown how the data structures perform when the time changes dynamically. The paper is concluded with some final remarks in section 7.
2 Related Work

Quality-of-service (QoS) in networks has been addressed for a variety of reasons. For some applications such as video and audio streaming it is required in order to avoid network jitter and congestions. 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 [2].

The requirement for advance reservations has been identified in several papers [3, 4, 7]. In addition to that, the relation between advance reservations on networks and advance reservations at the end-points of the network traffic, i.e. the computers involved, is described by [11]. 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 [5, 9]. In [6] 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 [8]. MPLS allows using constraint based routing and therefore is ideally suited to support traffic engineering using a bandwidth broker.

The admission control procedure in bandwidth brokers so far has not been studied in detail. The considerations presented in this paper are based on [10]. The segment tree described there is the foundation for our own implementations and tests. It was extended and studied further concerning both admission speed and especially memory consumption. However, our tests show arrays are better
suited in the environments considered.

3 Admission Control

![Diagram](image)

Figure 1: Advance Reservations and Book-ahead Time

The bandwidth broker 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 defined by the starting and finishing time and the peak bandwidth requirement which makes the problem of admission control two-dimensional: the bandwidth broker has to determine whether sufficient bandwidth is available for the given duration between starting and finishing time.

The data structures examined in this paper are implemented using *slotted time*. This means, the starting and finishing times have to be defined using a certain granularity, which cannot be changed during run-time. Choosing an appropriate granularity basically depends on the requirements of the application the bandwidth broker is used in. For the tests presented in this paper, a granularity of one minute was chosen. This seems to be sufficient for most types of reservations. Especially for advance reservations with a book-ahead of several weeks, it seems to be unlikely that finer granularity is required.
4 Data Structures

The data structures used for the evaluations are an array and a segment tree which was originally described in [10]. We made some extensions to the tree in order to improve the performance and to better suit the purposes of the prototype implementation of our bandwidth broker. In the following sections, the data structures and their properties are described.

4.1 Array

![slots to be checked](starting time) (finishing time)

Figure 2: Advance Reservation using an 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.

4.2 Segment Tree

The segment tree implementation was originally implemented by [10]. First of all, a short description of the data structure is given followed by the extensions we made.
Each node of the binary segment tree represents one period of the overall book-ahead time and the bandwidth reserved for that period. Both starting and finishing times are stored within each node together with two values related to the reserved bandwidth. The first value (denoted by \textit{node value}) is the bandwidth reserved during the particular period of time the node stands for. The second value (\textit{max value}) denotes the maximum sum of node values in any of the branches below the current node (see Figure 4). Starting at the top node, the two subsequent nodes at the level below each represent one half of the duration of their father node. In order to perform admission control, each reservation request is processed as follows:

1. Starting at the top node, each reservation whose period is covered by the current node, "falls" through to the next level below the current node, i.e. the admission procedure is done with the node at the level below which covers the particular period of the reservation.

2. If the duration of the request intersects with the duration of more than one node on the current level, the reservation is split at the intersection points and then each of the parts "falls" through, i.e. for each part the admission decision is made independently.

3. If a reservation or a part of it completely fits into the duration covered by
Figure 4: Segment Tree before and after Insertion

a tree node (such a node is called final node), the "fall-through" process stops for the respective part or the whole reservation\(^1\).

While the reservation or the parts of it "fall" through the tree, the node values of each tree node visited are added up (also for final nodes). When finally the process is completely stopped, i.e. all final nodes are found which completely cover the duration of a part of the original reservation, it is checked whether the requested bandwidth together with the computed sum does not exceed the link bandwidth. In case the bandwidth is sufficient, the request can be admitted. This has to be checked for each path from the top node to the final nodes. In

\(^1\)The final nodes are not necessarily leaf nodes.
case sufficient bandwidth is available for all parts of the reservation, the node values in the corresponding final nodes can be updated. Following that, each path has to be traversed backwards in order to update the max values in each node previously visited. Figure 4 shows a segment tree reflecting the situation shown in Figure 3 before and after a new request (bandwidth = 50, starting slot = 1, finishing slot = 6) is inserted, solid color denotes the final nodes.

4.2.1 Dynamic Memory Allocation

One of the major drawbacks of the segment tree is the memory consumption. Depending on the actual implementation, segment trees require up to 14 times more memory than arrays (see Section 6.4). Starting 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 can be reduced by only allocating the nodes that are actually required. This is done dynamically during run-time.

At the beginning, only the root node is present. In case a new reservation is added, missing tree nodes are generated on-demand during the tree traversal.

4.3 Worst-Case Memory Requirement

In the following, we assume a memory requirement of 4 bytes per integer and 4 bytes per pointer. When \( n \) is the total number of time slots covered, arrays require \( n \times 4 \) bytes. The original implementation of the segment tree uses 20 bytes per tree node. The memory requirement can be reduced by 8 bytes by not storing the start and stop times within the nodes but computing them during the traversal. In addition to that, using dynamic memory allocation it is possible to further reduce the requirement during the run-time of the broker. However, in the worst case there is still a requirement of \( (n \times 2 - 1) \times 20 \) bytes. Therefore,
arrays seem to have an advantage concerning memory consumption which can be proved in the following sections.

4.4 Dynamic Book-Ahead Time

![Diagram of two trees with book-ahead]

Figure 5: Dynamic Book-Ahead: Usage of two Trees

Another drawback of the segment tree is its unsuitability for dynamically advancing book-ahead time. In order to cope with that problem, two trees can be used as depicted in Figure 5. At the time the book-ahead window completely covers one tree, the other tree can be completely deleted. However, this does not reduce the worst-case memory consumption which is obviously doubled. In addition to that, in case a single reservation spans more than one tree this does also increase the admission time since reservations must be split and an admission decision has to be made in both trees, thus increasing the decision time. The amount of reservations that have to be split obviously depends on properties of the particular environment in which the bandwidth broker is implemented. In our tests, the additional decision time caused by split reservation requests was negligible (see Section 6.4).

Compared to this, arrays can be easily implemented as ring buffers using a single pointer to mark the current time (see Fig. 6). This does neither affect the admission speed nor the memory requirement of the array.
5 Test Environment

Firstly, the test environment is outlined followed by descriptions of the parameters used for the evaluations.

The tests were made on a 1 GHz Pentium III PC running Linux. The prototype bandwidth broker implementation is based on MPLS for traffic engineering, i.e. to influence the routing in the network. This allows to choose different routes in case a network path is completely utilized (see section 6.3).

Two types of tests were made. Firstly, the performance of only the single data structure (segment tree or array) was measured. Secondly the time used for the complete admission decision was examined using a network of 9 routers and 14 links. This allows to evaluate how the performance of the data structure influences the overall admission time of the bandwidth broker. In addition to the admission control task, in the second environment also the path computation in the network is included in the admission control procedure. In case a chosen link cannot suffice the bandwidth requirement of a given request, an alternative path must be found and checked. This means, before a reservation can be made on any of the links of a path, it is required to check each link on the path for sufficient bandwidth.

Therefore, admission control involves a two-phase procedure (obviously, the second phase is only required in case the first phase is successful):

1. *check* whether a reservation request can be fulfilled

2. *update* of the respective data structure
For the evaluation, we used two different sets of parameters for the reservations requests:

1. The same parameters as used in [10]: a look-ahead time of 30 days, slotted time using a granularity of five minutes, i.e. a total of 8640 slots, and reservation durations uniformly distributed in [20..180] minutes (i.e. 4 - 32 slots). The results show the superiority of arrays concerning both admission speed and memory requirement (see section 6.1).

2. The second set of parameters was introduced in order to compare the performance of both data structures under conditions where longer reservations are being made and both granularity and look-ahead is increased: a look-ahead time of 65536 minutes (i.e. approximately 1.5 months), a granularity of one minute, and reservation durations uniformly distributed in [10..2000] minutes.

So far, there is no indication of how reservation requests in reality might look like. Therefore it is not clear if these parameters reflect reality. However, the results presented in this paper allow to draw conclusions about the applicability of the data structures for any given set of reservation requests.

For both sets of parameters, the link bandwidth is assumed to be 100 MBit/s and the required bandwidth specified by each request is 64 kBit/s. As described in [10], we also used only the peak bandwidth to specify the resource requirement in a reservation request since other parameters such as jitter and delay are usually not considered in DiffServ environments.

In the following sections, we present the results of our evaluations showing that under nearly any condition, arrays are superior to segment trees concerning both memory requirement and admission speed.
6 Evaluation

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

6.1 Performance of the Data Structures

At first, the admission time of the data structures as a function of the duration of the reservation request was examined. Every single reservation request was made on an empty data structure, i.e. no other reservations were present.

![Figure 7: Admission Speed (left) and Memory Usage (right) as a Function of the Duration](image)

Figure 7 shows the results for the array and the tree implementation. It can be observed that arrays are competitive (compared to the tree implementation) 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. However, during the admission process, arrays have the advantage of constant access to the single slots whereas the tree implementation requires the traversal of several levels of the tree. The number of levels and therefore nodes visited during the admission decision is relatively high for short reservations since each recursion requires significant overhead.
The memory requirement of the data structures for a single reservation is also given in Figure 7. It can be observed that the data structures are relatively unaffected by the duration. Storing only a single reservation request, 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.

The tree implementation requires to allocate memory during the reservation request for every node visited during the admission process. Therefore the actual admission time for a reservation of given length might be lower in case these nodes are already present within the tree. 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.

6.2 Single-Link Admission Control

In this section, the admission time and memory consumption of the data structures for a single link are shown. The admission time includes the check phase and in case of success also the update phase (see Section 5). In the figures, each point represents the average of 1000 consecutive admission decisions. The data structures were evaluated using different parameters in order to determine admission speed and memory consumption.

First of all, we used the first set of parameters as described in Section 5. The results are shown in Figure 8. Since the duration of the reservation requests is below 180 minutes (i.e. 32 slots), it is not surprising (see Section 6.1) that the resulting admission times of arrays are significantly smaller than those of trees. Only, when the number of rejected reservations increases the difference
between array and segment tree closes substantially because only the check phase is required, see Section 6. 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 would be even wider.

In real-world scenarios, even longer reservation requests than previously considered might occur. Since arrays are very sensitive to changes of the duration (see Section 6.1), the second set of parameters was used for additional tests. The results are shown in Figure 9. It can be observed that compared to the tests with the first set of parameters, the relative difference of the admission
times between arrays and trees is decreased for successful reservations. The reason is the increased length of the requested reservations. However, the array implementation still requires significantly less admission time than the segment trees. This shows, the admission times of the array implementation also benefit significantly from rejected requests. Again, the memory consumption of the segment tree is much higher than that of the array, the gap between both data structures has widened.

6.3 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 results from Section 6.2 show that arrays are superior even using longer reservation requests, in this section the results were generated using only the second set of parameters. The first parameter set (which was used in [10]) is not considered since it results in extremely short reservations (i.e. between 4 and 32 slots) which can always be processed much faster using arrays. The source and destination node for a request was randomly chosen among the available nodes.

6.3.1 Path Calculation

The routing is done using Dijkstra’s shortest path algorithm DSP to find a route through the network. The path computation works as follows: a reservation is specified using the source and destination address together with the requested bandwidth, starting and finishing time. The DSP algorithm is used to compute a path in the whole network. Each link on the path is then checked for bandwidth. In case insufficient is detected for one or more links on the path, these links are
removed from the network (of course only for this route computation). After that, DSP is run again on the remaining network until finally either a path meeting the bandwidth constraint is found or there is no more path from source to destination. In the case of a failure of the bandwidth check a single-phase procedure would require to remove the current reservation from each link on the path where the reservation was successfully made. This shows, that it is necessary to use the two-phase admission procedure described in section 6 in order to avoid the additional processing time for removing requests from the data structures.

A different opportunity for calculating the path is to firstly compute the set of links with sufficient bandwidth and then to perform the DSP on the resulting network. However, since the bandwidth computation for each link requires significantly more processing time than the DSP algorithm, this would increase the total admission time. Since path calculation is a part of the admission process, the admission times shown in the following section contain a portion required for DSP.

Figure 10: Multi-Link: Admission Speed
6.3.2 Admission Speed

The admission times are shown in Figure 10. In general, the results are roughly the same as presented in Section 6.2: arrays have a significant advantage over segment trees (although a varying portion of the decision time is required for path calculation). 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.

6.3.3 Memory Usage

![Figure 11: Multi-Link: Memory Usage](image)

The memory usage of the data structures in the multi-link scenario shows the same properties as in the single-link scenario. Arrays require a constant amount of memory while the memory usage of our segment tree implementation varies over time. This does affect the memory usage of the whole network, since links with less utilization do not require considerable amounts of memory. However, our simulation shows, that even when some links are less utilized, the rest of the links have a higher utilization and therefore contribute more to the total memory requirement. Figure 11 shows, that the peak memory requirement of trees after
1 million requests exceeds the constant requirement of arrays by a factor of up to 10. Obviously, this amount of memory rises with the total number of links.

6.3.4 Influence of the Duration

In this section, we will show how the duration of the requested reservations affects the admission speed of arrays and segment trees.

<table>
<thead>
<tr>
<th>total requests = 100,000</th>
<th>array (avg. time)</th>
<th>segment tree (avg. time)</th>
<th>admitted requests (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>0.169417</td>
<td>0.251461</td>
<td>99.46</td>
</tr>
<tr>
<td>1000</td>
<td>0.201405</td>
<td>0.258621</td>
<td>98.72</td>
</tr>
<tr>
<td>1500</td>
<td>0.239161</td>
<td>0.253636</td>
<td>98.01</td>
</tr>
<tr>
<td>2000</td>
<td>0.293347</td>
<td>0.263389</td>
<td>95.43</td>
</tr>
<tr>
<td>2500</td>
<td>0.280302</td>
<td>0.231715</td>
<td>91.48</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>total requests = 1,000,000</th>
<th>array (avg. time)</th>
<th>segment tree (avg. time)</th>
<th>admitted requests (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>0.146593</td>
<td>0.204616</td>
<td>73.48</td>
</tr>
<tr>
<td>1000</td>
<td>0.146562</td>
<td>0.195943</td>
<td>50.05</td>
</tr>
<tr>
<td>1500</td>
<td>0.140825</td>
<td>0.193072</td>
<td>37.74</td>
</tr>
<tr>
<td>2000</td>
<td>0.137171</td>
<td>0.192729</td>
<td>30.17</td>
</tr>
<tr>
<td>2500</td>
<td>0.139345</td>
<td>0.198249</td>
<td>25.34</td>
</tr>
</tbody>
</table>

Table 1: Average Admission Speed for Different Average Reservation Durations

In Table 1, the average admission speed is shown for different durations of reservation requests. It can be seen, the admission speed using arrays depends heavily on whether a reservation is admitted or rejected. In general, segment trees are much slower than arrays in case of a rejection (see Section 6.3.2). Therefore, when the number of successful requests decreases, the average admission speed of arrays does not change significantly even in case the average duration of the requests rises. Consequently, in case most requests are successful the average admission speed of arrays can only keep up with the speed of trees up to an average duration of \( \approx 1600 - 1700 \).
6.4 Dynamic Book-ahead Time

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 run-time of the bandwidth broker, the book-ahead time advances and the data structures must be able to deal with this in an appropriate way. As described in section 4.4, arrays can be easily adapted for the dynamic scenario by implementing them as ring buffers with a pointer to the current time. This does not lead to additional admission time or memory consumptions. In contrast to this, when using the segment tree structure, two trees are required.

![Image](image.png)

Figure 12: Admission Speed (left) and Memory Usage (right): dynamic book-ahead

The effects of the advancing time on admission speed and memory usage can be seen in Figure 12. The results for the dynamic 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. This shows the effect of dynamic book-ahead time over a longer period. 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. The reservation requests were generated according to the second set of parameters as described in Section 6.2, using multi-link admission control.

Compared to the admission speed in the static case, the decision time in-
creases towards the end of the reservations (i.e. starting approximately with request # 750000) in the dynamic scenario. Although only a few reservations are split and suffer from longer admission times using the segment tree, it has to be taken into account that in the static scenario the number of rejected requests rises faster with the total number of requests. This leads to reduced decision times (only the check phase must be performed). Whereas in the dynamic scenario, the number of rejected requests rises slower since the time pointer advances rather rapidly (after each 10 admission requests). This means, a lot more requests can be admitted which also reduces the admission speed because in those cases the check and the update phase are required.

Figure 12 also shows the effect of dynamic book-ahead time on the overall memory requirement of the trees: compared to the memory usage of just a single tree about 60 % more memory is required, i.e. about 14 times more than required by an array. 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.

7 Conclusion and Future Work

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 nearly any scenario. The experiments described in this paper show, that arrays are inferior
when only few reservations are rejected and the average duration of admitted reservation requests is rather long, i.e. significantly above 1000 slots. Future work might focus on whether it is possible to decrease the slot granularity for requests with starting and finishing times near the end of the book-ahead time, which would then lead to a better performance of arrays. Moreover, in case only requests with a long reservation duration can be assumed, the overall granularity can be decreased. The reason is, that it might be not important to define starting and finishing with a granularity of one minute in case the transmission e.g. lasts for a week.

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 a long duration can be handled efficiently using arrays because rejected requests require only a short decision time.

The memory requirement of arrays is always significantly below that of segment trees, especially in the dynamic scenario where even two trees are required resulting in nearly doubled the overall memory consumption.

References


