

An Efficient Data Structure for Advance Bandwidth Reservations on the Internet

Andrej Brodnik^{*†}

Andreas Nilsson[†]

May 19, 2002

Abstract

In this contribution we present a problem of resource reservation during some time. We show that the problem has a lower bound of $\Omega(\log n)$ per operation on average and also give a matching upper bound algorithm.

1 Introduction

In Computer Communications we need to make a reservation of bandwidth over the Internet to provide *Quality of Service (QoS)* for the end users. The IETF (Internet Engineering Task Force) defined a standard for Integrated Services in routers ([10, 21]) and the end-to-end reservation setup protocol RSVP ([6]). The protocol had a flaw that it did not scale well ([14]). Therefore IETF came up with a new approach, known as *differentiated services (diffserv)*, [4]. Olov Schelén et. al. ([17, 18]) used the *diffserv* to design a new QoS architecture. In this architecture they provide virtual leased lines using the differentiated services to perform admission control through the system of agents. The agents work on per-hop basis and they need to maintain a database of the reservations made on their hop. In the backbone of the Internet it will most likely be many reservations to administrate and hence the use of an efficient data structure will be required. Moreover, in the design of the agents the authors propose that a single agent administrates several hops to make it more attractive for the ISP's (Internet Service Provider). Such a scenario puts even bigger need for use of an efficient data structure.

The bandwidth reservation problem is a special case of a more generic problem, where we need to administrate a limited resource over the time; e.g. use of human resources, computational power of super-computer, pool of cars etc. Although the solution in this paper covers all these problems, we use the term *bandwidth* when we talk about the reserved resource.

Definition 1 *In the Bandwidth Reservation Problem we have a fixed amount of bandwidth to administer. Customers want to make reservations $R = \{B, I\}$ for a part of the bandwidth B during a time interval $I = [t_0, t_1]$ ($t_0 < t_1$, the interval starts at time t_0 and ends at time t_1 , and it includes both end points). The operations to support, besides initialization and disposition, are:*

- **Reserve**(B, I), that reserves B units of bandwidth for the time period $I = [t_0, t_1]$, where $t_0 \leq t_1$.
- **Free**(B, I), that frees the bandwidth B during the interval I . Note that freeing the bandwidth is the same as making a reservation with a negative bandwidth.
- **MaxReserved**(I), that returns maximum reserved bandwidth during the interval I .

For the sake of clarity, we sometimes use the subscripts q for queries and r for reservations. For example, a reservation interval $I_r = [t_{r0}, t_{r0}]$. In the paper we also use the notation $\max(x, y)$ denoting a function returning the bigger of x and y .

In the literature we could not find any references to the bandwidth reservation problem where the points in time, when the reservation can be made, are arbitrary (not predefined). However, the problem is similar to problems we find in other fields of computer science that handle intervals on a real line (e.g. computational geometry, dynamic

^{*}Department of Theoretical Computer Science, Institute of Mathematics, Physics, and Mechanics, Ljubljana, Slovenia

[†]Department of Computer Science and Electrical Engineering, Luleå University of Technology, Luleå, Sweden

computation and geometric search [1, 2, 7, 8, 15]). These problems are generally solved using *segment trees* ([16, 19]), which were introduced by Bentley ([3]) as a solution to the Klee’s rectangle problem ([12]). The limitation in all these problems is that the end-points of the intervals belong to a *fixed set* of points while in our problem the set of points changes during the time.

Kuchem et. al. in ([13]) presented in a way similar data structure to ours, although it still deals with a fixed set of points. They use the structure in a VLSI design. Bose et. al. independently developed in ([5]) a similar data structure to solve a number of geometric problems.

Another pair of related problems are the well studied partial sum problem ([9], brief in [11]), and the prefix sum problem ([9]). In the prefix sum problem we have an array $V(i), 1 \leq i \leq n$ on which we want to perform these two operations: (1) `Update(i, x): $V(i) = V(i) + x$` ; and (2) `Retrieve(m): $\sum_{k=1}^m V(k)$` for arbitrary values of i, x and m . In [9] Fredman shows a lower bound of $\Omega(\log n)$ for the problem under the comparison based model. In the same paper Fredman also presents an algorithm with a matching upper bound.

In the rest of the paper we first show that the logarithmic lower bound carries over to the bandwidth reservation problem. We continue with a presentation of a data structure we call *BinSeT* (binary segment tree) that gives us a matching upper bound. We conclude the paper with final remarks.

2 Lower bound

Theorem 1 *The problem of bandwidth reservation requires at least $\Omega(\log n)$ comparisons per operation on the average, where n is a number of intervals we are dealing with.*

Proof: Assume that we have a solution to the bandwidth reservation problem that requires $o(\lg n)$ time. We will show how to use such a solution to solve the prefix sum problem in time $o(\log n)$ which contradicts the lower bound by Fredman ([9]).

We translate the array of elements in the prefix sum problem into end-points of intervals. More precisely, the $V(i)$ element of the array is represented by the interval that starts at point i and ends at the right most point n : $[i, n]$. Therefore, the reserved bandwidth at point p is the sum of all reserved bandwidths for intervals starting at points j , where $1 \leq j \leq p$. This gives us the following translation of prefix-sum problem operations:

- the operation `Update(i, x)` into `Reserve($x, [i, n]$)`; and
- the operation `Retrieve(j)` into a query `MaxReserved($[j, j]$)`.

This translation gives us an $o(\lg N)$ solution to the prefix sum problem and hence contradicts the lower bound by Fredman. *QED*

Note that, the prefix sum as presented by Fredman ([9]) is also a *static problem* – i.e. the array of elements neither expands nor shrinks. On the other hand, the solution we present in the following section does support insertion of new points (intervals) and deletion of points (intervals). Hence, by using the translation in the proof we also get a logarithmic solution to the dynamic version of the prefix-sum problem.

3 Upper bound

To prove an upper bound we present a data structure called *BinSeT* that supports the required operations in logarithmic time. Before going into detail presentation we describe how we represent the reservations.

3.1 Representation of Intervals

We do not represent an reservation interval as a single entity, but we split it into two, what we call, *reservation events*. A reservation event is point in time when an increase or decrease in the amount of a reserved bandwidth occurs. For example, we store a reservation $R = \{B, [t_0, t_1]\}$ as reservation events $E_0 = (t_0, +B)$ and $E_1 = (t_1, -B)$. In other terms, we convert an interval $[t_0, t_1]$ into two semi-infinite intervals $[t_0, +\infty]$ and $[t_1, +\infty]$.

In the rest of the paper we will describe a data structure that handles the reservation events, or semi-infinite intervals, efficiently. The operations mentioned in Definition 1 are converted into:

- `Reserve($B, [t_0, t_1]$)` into insertion of reservation events $E_0 = (t_0, +B)$ and $E_1 = (t_1, -B)$; and

- $\text{Free}(B, [t_0, t_1])$ into insertion of reservation events $E_0 = (t_0, -B)$ and $E_1 = (t_1, +B)$.

If we want to store extra information with each reservation we introduce an additional dictionary data structure to store this information and bind the reservation events to records in the dictionary.

3.2 Data Structure

The binary segment tree BinSeT is a data structure that combines properties of a binary and a segment tree. The former gives us possibility to dynamically insert and delete reservation events and the later let us answer the queries about the maximum reserved bandwidth. In detail, the leaves represent and store information about the reservation events, while each internal node covers a segment (interval) I and stores an information about the values (bandwidth) on that interval. We have the following invariance:

Invariance 1 *The information stored with the interval I is the maximum value μ on the interval and the change δ of the value on the interval.*

Besides we split the interval into two sub-intervals in such a way that the number of reservation events in each of them differs for at most one. Such a split gives us the necessary and sufficient condition to get a balanced data structure. Obviously, the internal node also stores the pointers to the nodes representing sub-intervals and the splitting time τ . The splitting time τ is in fact the start time t_0 of the right sub-interval.

The detail data structure is represented in Algorithm 1. The structure is slightly different from the one described

```
typedef struct _sBinSeT {
    tResource  $\mu$ ;
    tResource  $\delta$ ;
    tTime  $\tau$ ;
    struct _sBinSeT* left;
    struct _sBinSeT* right;
} tBinSeT;
```

Algorithm 1: Binary segment tree definition.

in the previous paragraph since it does not store times t_0 and t_1 , but only the τ . The structure, as it will be seen in the description of operations, still fulfills all the requirements. At this point we note two things: first, a node has either two sub-trees (an internal node) or none (a leaf); and second, a leaf stores in δ and in μ amount of the reserved bandwidth at the reservation event it represents and in τ the time of the event. As a consequence of the first observation we conclude, that the number of internal nodes is one less than the number of leaves. Since the number of leaves is at most $2n$, where n is a number of reservation intervals, this proves the following lemma, under the RAM model:

Lemma 1 *The size of the BinSeT storing n reservation intervals is $\Theta(n)$ words.*

3.3 Operations

Finally we describe how to implement operations from Definition 1. All our solutions will be recursive and will start traversing the data structure from the root. We assume that we store with BinSeT also the time of the first (t_f) and the last (t_l) reservation event. These are also times t_0 and t_1 , respectively, for the root of the complete BinSeT. If we descend in the left subtree, then the t_0 and t_1 for this subtree become values t_0 and τ , respectively, of the root. We treat similarly the right subtree. This is also the reason why we need not store values t_0 and t_1 with a node.

We start with a query `MaxReserved`. Assuming Invariance 1 we prove:

Lemma 2 *Let BinSeT be balanced (cf. [20]) then there exists a $O(\log n)$ worst case implementation of query `MaxReserved`.*

Proof: The correctness of the proof uses induction. Due to the limited presentation space we give only a justification of the induction step. Let the query be for the interval $I_q = [t_{q0}, t_{q1}]$ and let the node cover interval $[t_0, t_1]$. If $t_{q0} = t_0$ and $t_{q1} = t_1$, the answer is μ of the node. If $t_{q1} \leq \tau$ ($\tau \leq t_{q0}$) then the answer is the

same as answer to the query in the left subtree l , $\text{MaxReserved}(l)$ (right subtree r to which we add $l.\delta$, $l.\delta + \text{MaxReserved}(r)$). In the third case when $t_0 \leq t_{q0} < \tau$ and $t_{q1} = t_1$ ($t_0 = t_{q0}$ and $\tau < t_{q1} \leq t_1$), the answer is $\max(\text{MaxReserved}(l), (l.\delta + r.\mu))$ ($\max(l.\mu, l.\delta + \text{MaxReserved}(r))$). Finally, the fourth and the most general case is when $t_0 < t_{q0} < \tau < t_{q1} < t_1$. Here the answer is

$$\max(\text{MaxReserved}(l), l.\delta + \text{MaxReserved}(r)) . \quad (1)$$

To see that the running time of the query is logarithmic, i.e. proportional to the height of the BinSeT, observe that the fourth case occurs only once. *QED*

Next we describe insertion, but due to lack of space we leave out the detail description of the deletion. Note that the deletion is always implicit when the amount of reserved bandwidth at a leaf becomes 0.

Lemma 3 *There exists $O(\log n)$ worst case implementation of updates Reserve and Free in BinSeT.*

Proof: As explained in sect. 3.1 each of operations Reserve and Free is converted into a pair of insertions of reservation events. We will show that the reservation event insertion takes logarithmic time which will prove the lemma.

The insertion starts at the root and recursively descends to the leaves. We descend the BinSeT using τ and time of the reservation event t_R to find the proper leaf l . When we reach the leaf its τ can be the same as t_R of the inserted event. If so, we add the values. At this point the value can become 0 and we delete the leaf and replace its ancestor with leaf's sibling. If the leaf's t_0 is different from t_R , we insert a new internal node with $\mu = \delta = l.\mu$, τ either $l.\tau$ or t_R as appropriate, and make an old leaf as one of its leaves. Besides we create a new leaf with $\tau = t_R$ and $\mu = \delta = B_R$.

In the last phase we climb back up to the root and at nodes appropriately update values μ and δ . Values δ are always increased for B_R . on the other hand the value of μ is set to $\max(l.\mu, l.\delta + r.\mu)$, similarly as in eq. (1).

The tree obtained this way might be unbalanced, but we can re-balance it in a logarithmic time (cf. AVL trees, [20]). Due to lack of space we leave proof to the reader. *QED*

Putting all results together brings us to the final theorem:

Theorem 2 *There exists a solution to the Bandwidth Reservation Problem under the comparison based machine model that requires $\Theta(\log n)$ per operation and $\Theta(n)$ words of space. This is tight.*

Obviously it is straight forward to adapt the solution to handle also queries of the minimum reserved bandwidth. Moreover, using the translation in Theorem 1 we also get a logarithmic time solution to the dynamic versions of partial sum problem and of prefix sum problem.

4 Conclusions

We showed that the data structure BinSeT (binary segment tree) solves the dynamic version of the Bandwidth Reservation Problem optimally (space- and time-wise) under the comparison based model. The solution requires $\Theta(\log n)$ time for the queries and updates and $\Theta(n)$ space. This is a substantial improvement over the previous solution which was static only – new intervals could not be inserted.

There are a number of open problems left. For example, what are lower and upper bounds under the cell probe model and bounded universe?

References

- [1] Andrew Rau-Chaplin Albert Chan, Frank K. H. A. Dehne. Coarse-grained parallel geometric search. *Journal of Parallel and Distributed Computing*, 57(2):224–235, 1999.
- [2] L. Arge. The buffer tree: A new technique for optimal i/o-algorithms, 1994.
- [3] J. L. Bentley. Algorithms for Klee's rectangle problems. 1977.
- [4] S. Blake et al. An architecture for differentiated services. RFC (Informational) 2475, IETF, December 1998.

- [5] P. Bose, M. van Kreveld, A. Maheshwari, P. Morin, and J. Morrison. Translating a regular grid over a point set. *Computational Geometry: Theory and Applications*. Accepted for publication.
- [6] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reservation protocol (RSVP) – version 1 functional specification. Internet Draft, Internet Engineering Task Force, November 1996. Work in progress.
- [7] Chan, Dehne, and Rau-Chaplin. Coarse grained parallel next element search. In *IPPS: 11th International Parallel Processing Symposium*. IEEE Computer Society Press, 1997.
- [8] David Eppstein. Dynamic three-dimensional linear programming. In *IEEE Symposium on Foundations of Computer Science*, pages 488–494, 1991.
- [9] Michael L. Fredman. The complexity of maintaining an array and computing its partial sums. *Journal of the ACM*, 29(1):250–260, January 1982.
- [10] R. Guerin, C. Partridge, and S. Shenker. Specification of guaranteed quality of service. Request for Comments (Proposed Standard) 2212, Internet Engineering Task Force, October 1997.
- [11] Thore Husfeldt and Theis Rauhe. Hardness results for dynamic problems by extensions of fredman and saks’ chronogram method. In *Proc. 25th Int. Coll. Automata, Languages, and Programming*, number 1443 in Lecture Notes in Computer Science, pages 67–78. Springer-Verlag, 1998.
- [12] V. Klee. Can the measure of $\bigcup_1^n [a_i, b_i]$ be computed in less than $O(n \log n)$ steps? *Amer. Math. Monthly*, 84:284–285, 1977.
- [13] R. Kuchem, D. Wagner, and F. Wagner. Optimizing area for three-layer knock-knee channel routing. *Algorithmica*, 15(5):495–519, May 1996.
- [14] A. Mankin, F. Baker, B. Braden, S. Bradner, M. O’Dell, A. Romanow, A. Weinrib, and L. Zhang. Resource reservation protocol (RSVP) – version 1 applicability statement and some guidelines on deployment. RFC (Informational) 2208, IETF, September 1997.
- [15] K. Mehlhorn and F. P. Preparata. Routing through a rectangle. *Journal of the ACM*, 33(1):60–85, 1986.
- [16] Kurt Mehlhorn. *Data structures and algorithms 3: Multi-dimensional searching and computational geometry*. Springer-Verlag, 1984. 91-032.
- [17] O. Schelén and S. Pink. An agent-based architecture for advance reservations. In *IEEE 22nd Annual Conference on Computer Networks (LCN’97)*, Minneapolis, Minnesota, November 1997.
- [18] O. Schelén and S. Pink. Sharing resources through advance reservation agents. In *Proceedings of IFIP Fifth International Workshop on Quality of Service (IWQoS’97)*, New York, May 1997.
- [19] M. I. Shamos and F. P. Preparata. *Computational geometry*. Springer, 1985.
- [20] Ronald L. Rivest Thomas H. Cormen, Charles E. Leisserson. *Introduction to Algorithms*. The MIT Press, McGraww-Hill Press, 1990.
- [21] J. Wroclawski. Specification of the controlled-load network element service. Request for Comments (Proposed Standard) 2211, Internet Engineering Task Force, October 1997.