Darmstadt University of Technology

The Token Bucket Allocation and Reallocation Problems

(MPRASE Token Bucket)

Oliver Heckmann, Frédéric Rohmer, Jens Schmitt
heckmann@kom.tu-darmstadt.de
rohmerf@yahoo.fr
schmitt@kom.tu-darmstadt.de

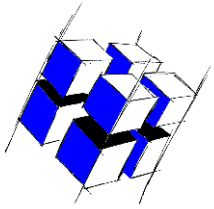
KOM Technical Report 12/2001
Version 1.0
December 2001

Last major update 30.12.01
Last minor update 13.01.02

Industrial Process and System Communications (KOM)

Department of Electrical Engineering & Information Technology
Merckstraße 25 • D-64283 Darmstadt • Germany

Phone: +49 6151 166150
Fax: +49 6151 166152
Email: info@KOM.tu-darmstadt.de
URL: http://www.kom.e-technik.tu-darmstadt.de/



Abstract

This paper deals with the dimensioning of token buckets. Two related problems are formulated and solved. The first is the token bucket allocation problem (TBAP) which is finding the cost-minimal token bucket for the transmission of a given VBR traffic stream, e.g. an MPEG movie, from the user's point of view. This problem has been treated in literature before but as we will show not completely. We will derive and efficient and exact algorithm for this problem. As a second step, the token bucket **re**allocation problem (TBRP) is investigated based on the results for the TBAP. The dynamic token bucket dimensioning problem consists of finding the cost-minimal *series* of token buckets for the transmission of the stream. For this problem an exact algorithm is again presented and furthermore, several heuristics are devised. The best heuristic comes closer than 0.25% to the results of the exact algorithm and is orders of magnitudes faster as several numerical simulations show. We also try a mulitregression analysis for token bucket dimensioning.

This paper fits into the MPRASE (Multi-Period Resource Allocation at System Edges) framework, the treated problems are single-customer, single-provider problems with a two-dimensional resource model (the token bucket), a linear cost model, and a deterministic edge.

1 Introduction

In order to be able to offer any kind of Quality of Service (QoS) guarantees traffic has to be regulated. Traffic shapers and policers are common elements in both IntServ [2] and DiffServ [1] networks. Token buckets are the most popular traffic regulating mechanism, especially as they are easy to implement, see e.g. [14, 11, 12, 32, 28] for the role of token buckets in a DiffServ environment.

We look at a traffic flow, e.g., the transmission of an MPEG video, that is going to be sent and that will be token bucket regulated. For a given traffic stream there is an infinite number of token buckets the stream complies to as there is a certain tradeoff between r and B. A certain decrease in B can be compensated by an increase in r. From the user's point of view the following question has to be answered: What is the cost-minimal token bucket that a traffic stream complies to?

That is the first of two problems that this paper discusses, token bucket allocation problem (TBAP): A single token bucket has to be dimensioned for a flow that is known in advance as when streaming a pre-recorded video from a server towards a client. We assume that the allocation of the token bucket imposes certain (real or fictive) costs. Our aim is to find the cost-minimal token bucket.

To some extent this problem has already been discussed in literature:

According to [33] the first work to efficiently calculate the minimal bucket depth of a token bucket for a given token rate - and that is a subproblem of the STBD - was done by Partridge and Garrett in 1994 [24]; their algorithm Send-Now is also described in [33]. An algorithm for the same problem that is more flexible as it does not rely on a full bucket in the first period is derived in [33]. That algorithm was also used in this work. Both mentioned papers also deal with calculating the minimal bucket depth for a given rate when a certain queue is added before the token bucket in which the stream can be hold while it is waiting for enough tokens to be accumulated.

However, these works look at the optimal bucket depth for a given rate but do not calculate the optimal rate.

Keshav [17] proposes as a heuristic for token bucket dimensioning to choose the "knee area" that the $B_{opt}(r)$ curve shows, outside which small changes in rate resp. bucket depth can only be compensated by greater changes in the other parameter. However, Keshav does not propose a cost function with which the preference of rate and bucket depth can be weighted and he proposes no algorithm to find the area. Also other works [26] show that the "knee area" is not straightforward to find for long range dependent traffic.

In this paper, after the token bucket and a cost model for it are explained in Section 2, we present an exact and efficient algorithm that calculates the optimal rate and optimal bucket depth for a given traffic trace in Section 3. After that we look at a second more complex problem, the token bucket *re*allocation (TBRP) problem. Instead of using one token bucket for the transmission a *series* of token buckets could be used. Some systems explicitly support the renegotiation of parameters (IntServ RSVP [41]), others were extended to support renegotiation (ATM [40]). In Section 5, we analyse and compare the performance of the algorithms for the TBRP. We next try a multiregression analysis for token bucket dimensioning before discussing related work in Section 7, and drawing conclusions in Section 8.

2 Token Buckets

This chapter is borrowed mainly from [8] and describes token buckets for continuous and discrete streams. Also a cost model for token buckets is discussed.

2.1 Continuous Model

Token Bucket characteristics One common way to describe the bandwidth and burst characteristics of traffic sources is the token bucket, which is described by two parameters: the token rate *r* and the bucket depth *B*. It works as follows:

To be allowed to send *n* data units (e.g. bytes or packets) the sender must own *n* tokens. The sender starts with a certain number of tokens in his bucket and accumulates them at a rate of *r* per second. However he can't aquire more tokens than the bucket depth *B* at any time. The token bucket enables the sender to send a burst of length *B* as fast as he wants (as fast as his hardware is able to), but over a sufficiently long interval, he can't send more than *r* bytes rsp. packets per second.

A common way [20, 5, 21] to picture the token bucket characteristics in continuous time is depicted in figure 1. The *y* axis is the total number of bytes sent since a point in time t_0, the *x* axis is the time since t_0 . A valid flow (it's arrival curve) must remain below the depicted curve that starts at *B* and has a slope of *r* at all times $t > t_0$.

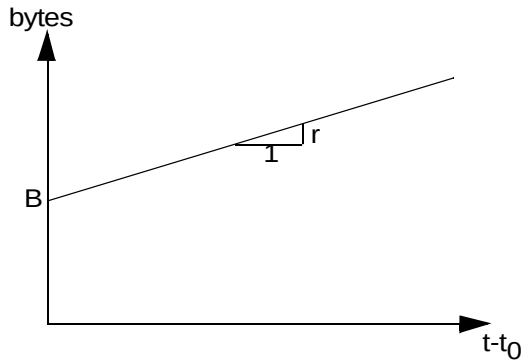


Figure 1: Token-Bucket curve

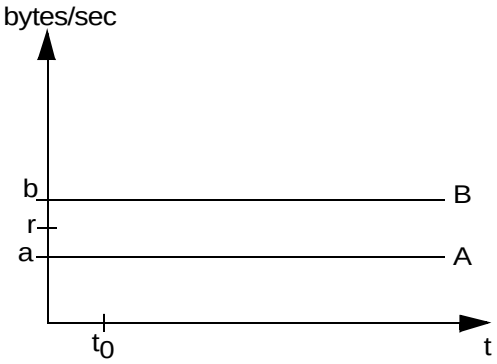


Figure 2: Two example data flows

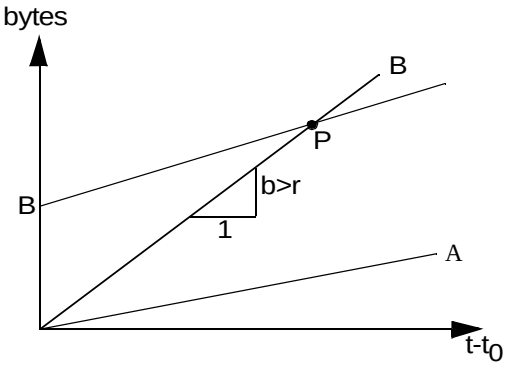


Figure 3: Flow B running out of tokens at point P

Fig. 2 shows two example flows with constant data rate. Note that the *y* axis here shows the bandwidth (bytes / second) and not the aggregated number of bytes as in figure 1. A transformation[1] of fig. 2 into fig. 1 is shown in fig. 3 for a chosen time t_0. As we can see, flow *A* remains below the token-bucket curve and so has enough tokens all the time while flow *B* is sending too much data and running out of buckets at point *P*.

[1] The transformation is simply an integral over the time starting at t0.

2.2 Discrete Model

For modelling puposes we use discrete time intervals. Fig. 1 corresponds to Fig. 4, rate *r* refers to one period now (instead of one second). Tokens are filled into the token bucket with the following algorithm in each period *t*:

1. The sender gets *r* new tokens.
2. In order to send data he pays one token per data unit (e.g. packet or byte). The maximum amount of data he can send in this period is the number of tokens he has left in his bucket plus the new tokens. The theoretically possible maximum burst is therefore $B + r$.
3. The unused tokens are stored in the bucket up to the bucket depth *B*, surplus tokens are lost.

The questions remains with how many tokens the bucket is filled in the first period. In order to remain general we assume the following in this paper: An additional constant δ $(0 \le \delta \le 1)$ is added, the bucket is filled with $\delta \cdot B$ tokens at the beginning of the first period. We assume that this parameter δ is fixed (by the provider); it is not a variable the user can set.

Figure 5 shows two example flows that are transformed for the critical interval t_0 (see figure 6). As we can see, flow *C* equals the token-bucket curve while flow *D* remains below it for all the time.

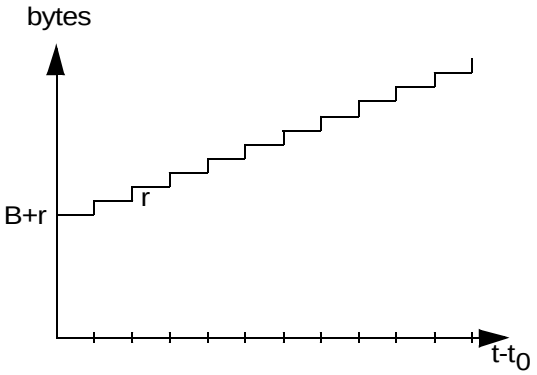


Figure 4: Token-Bucket curve in discrete time

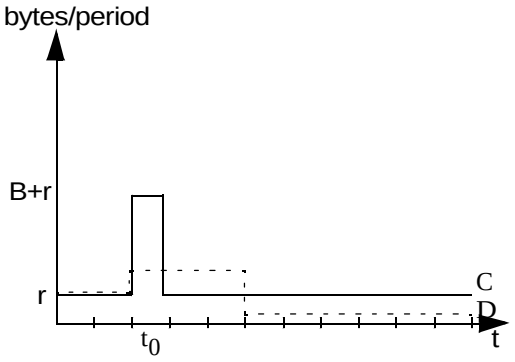


Figure 5: Two example data flows in discrete time

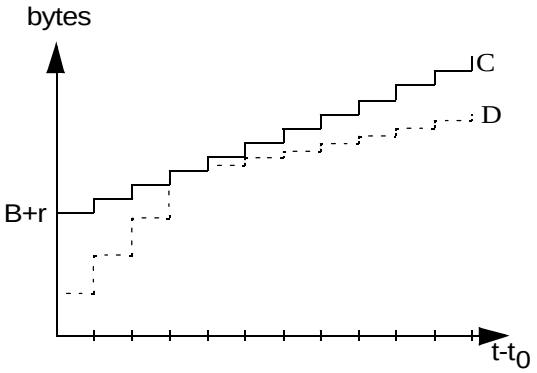


Figure 6: Aggregation of the two data flows

2.3 Pricing Token Buckets

Let's think about arguments for linear prices for the rate and the bucket depth. It is generally reasonable to assume linear prices for bandwidth, because otherwise it would be possible to arbitrage the provider by buying large quantities and reselling them in smaller quantities or by buying small quantities and reselling them in larger quantities. This is why we can reasonably assume linear prices for rates.

But is it also reasonable to assume linear prices for the bucket-depth? It is for the following kind of contract:

“The provider is offering some kind of guaranteed service, that means he is guaranteeing that none of packets are dropped (as long as they comply with the token-bucket model) and every burst is served within a certain time Υ.”

If accepting a token bucket flow, the provider must reserve some bandwidth for serving that flow. The amount of necessary bandwidth depends on Υ as is shown in figure 7.

To guarantee Υ, the provider must be ready to serve the token-bucket flow with a rate R which is proportional to B and independent of the token-bucket rate r.

$$\Upsilon = \frac{B}{R} \Rightarrow R = \frac{B}{\Upsilon}$$

So B translates linearly into a bandwidth R. As we have argued, it is sensible to assume linear prices for bandwidths, so it is also sensible to assume linear prices for the bucket depth, too.

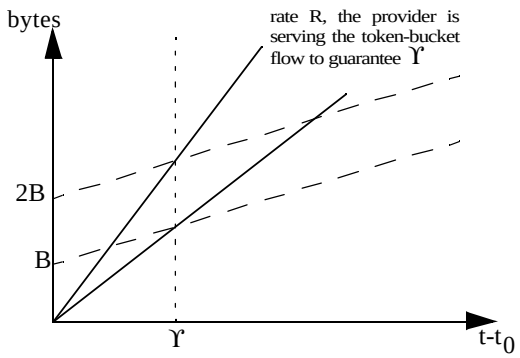


Figure 7: Bucket depth and needed bandwidth

2.4 Deterministic Knowledge

In this paper we also assume the sender acts without uncertainty. That means he knows his coming needs before he starts communicating. This assumption is valid e.g. for video playback from a disc. Moreover, it makes us able to concentrate on the deterministic problems first and later adapt them to uncertainty. MPRASE problems under uncertainty are discussed e.g. in [30].

3 The Token Bucket Allocation Problem (TBAP)

In this section the token bucket allocation problem (TBAP) and some exact algorithms to solve it are presented. The TBAP is a simplified version of the single provider token bucket allocation problem as described in [8] and the token bucket **re**allocation problem (TBRP) described in Section 4.

A client wants download or send real-time data (e.g. a movie) and has to specify the two token bucket parameters (r, B) for the flow x_t $(t = 1, ..., T)$. The two parameters are chosen before the transmission and cannot be changed afterwards. For the calculation of the parameters we assume deterministic knowledge about flow, the amount of data that has to be transmitted x_t each period t is known beforehand as it would be when transmitting a video from a hard disc. The bucket is supposed to contain $\delta \cdot B$ tokens at the beginning ($\delta \in [0, 1]$), δ is given. Our aim is to find an algorithm which finds the cost minimal token bucket (r, B). In order to do that we first have to have a cost model for the token bucket. This is discussed next in 3.1. Then we formulate the TBAP as a MIP problem in 3.2. We then strive for other algorithms to solve the TBAP by first looking at a subproblem that is finding the optimal B for a given r in 3.3. With this knowledge we can then in the last section of this chapter show an efficient algorithm for the TBAP.

3.1 A Cost Model for the TBAP

The costs model of the TBAP can only depend on four parameters: r, B, δ, and the number of periods T. Because T and δ are given, the cost-minimal pair (r, B) has to be found.

As explained in Section 2.3 it is reasonable to assume linear costs both for r and B. Introducing C_r and C_B as cost coefficients we define the costs per period P for a token bucket with the parameters (r, B) to be $P = C_r \cdot r + C_B \cdot B$.

3.2 The Mixed Integer Programming Approach to the TBAP

3.2.1 Formulation of the TBAP as a MIP-Problem

Our problem can be described as a MIP-Problem (Mixed Integer Programming Problem, [13]) as follows:

Variables:

- y_t Tokens in the bucket at the beginning of the period $t = 1,...,T+1$
- B Bucket depth
- r Token Rate

Parameters:

- x_t Data to be transmitted in period $t = 1,...,T$
- C_r Rate costs, costs per reserved capacity-unit r
- C_B Bucket depth costs, costs per bucket depth unit B
- δ Bucket starting factor ($\delta \in [0, 1]$), see Section 2.2

$$\textit{Minimize} \quad C_r\ r + C_B\ B \tag{1}$$

subject to

$$y_1 \le \delta \times B \tag{2}$$

$$y_t \le B \qquad \textit{for all } t = 2,...,T \tag{3}$$

$$y_t \leq y_{t-1} + r - x_{t-1} \qquad \text{for all } t = 2,...,T+1 \qquad (4)$$

$$y_t \geq 0 \qquad \text{for all } t = 1,...,T+1 \qquad (5)$$

$$r, B \geq 0 \qquad (6)$$

The target function (1) minimizes the costs, constraint (2) sets the number of tokens for the first period. (3) makes sure that there are never more than B tokens in the bucket, (4) accounts for the new tokens gained (r) and the ones spent (x_{t-1}) while (5) and (6) are the non-negativity constraints for the variables.

3.2.2 Performance Evaluation

It is possible to use standard MIP solving techniques like Branch and Bound with LP Relaxation to solve this problem [13]. But the costs of such an algorithm in computation time and especially in memory are quite high as can be seen in Table 2. Problems with T = 1000 take more than a second to solve and problems with T > 10'000 could not be solved on a 700MHz Pentium III Processor with 256 Megabytes RAM using the commercial MIP solver CPlex by Ilog [15], see Section 3.4.5.

Because of this we try to find other algorithms to solve this problem.

3.3 An efficient algorithm for the calculation of *B* for a given *r*

Before we derive an algorithm for the calculation of the cost minimal token bucket (r, B) we first develop and explain an algorithm which calculates the optimal (minimal) B for a given r. This algorithm and most of the mathematical properties in this part are directly inspired by [33]. We need this algorithm later to derive the cost minimal pair (r, B).

3.3.1 Relationship between the optimal *B* and *r*

We first describe how B and r are related. With the same notations as before we can write:

$$y_1(r, B) = \delta B \qquad (7)$$

$$y_t(r, B) = min((y_{t-1}(r, B) + r - x_{t-1}), B) \qquad \text{for all } t = 2,...,T \qquad (8)$$

A set of necessary and sufficient conditions for the token-bucket parameters to allow the transmission of all data is:

$$(y_1 + r = \delta B + r) \geq x_1 \qquad (9)$$

$$(y_t + r = min((y_{t-1} + r - x_{t-1}), B) + r) \geq x_t \qquad \text{for all } t = 2,...,T \qquad (10)$$

(9) and (10) form a system of n conditions . The last of them can be rewritten as

$$(y_{T-1} + 2 \times r - x_{T-1}) \geq x_T \qquad (11)$$

$$\text{and } (B + r) \geq x_T \qquad (12)$$

Combining (11) and (10) for $t = T-1$, we get:

$$(y_{T-1} + r = min((y_{T-2} + r - x_{T-2}), B) + r) \geq x_{T-1}, x_T + x_{T-1} - r \qquad (13)$$

Hence, we obtain:

$$B \geq x_{T-1} - r, x_T + x_{T-1} - 2r \qquad (14)$$

and

$$y_{T-2} \geq x_{T-1} + x_{T-2} - 2r,\ x_T + x_{T-1} + x_{T-2} - 3r \qquad (15)$$

Continuing this process inductively for $t = T-2,...1$, we obtain the explicit solution for (9) and (10)

$$B \geq \sum_{i=u}^{v} x_i - r(v - u + 1) \qquad \text{for } 2 \leq u \leq v \leq T \qquad (16)$$

$$\delta B \geq \sum_{i=1}^{v} x_i - rv \qquad for\ 1 \leq v \leq T \tag{17}$$

For $\delta = 0$ (17) doesn't restrict B but implies

$$r \geq \max_{1 \leq v \leq T} \left(1/v \times \sum_{i=1}^{v} x_i \right) \tag{18}$$

This leads to the following theorem:

Theorem 1: (Minimal *r*)

For $\delta = 0$ the minimum *r* is given by (18), for $\delta > 0$ the minimum *r* is zero.

Proof: The first part of the theorem was shown above, for $\delta > 0$ it is obvious that *r* can be zero if

$$B \geq \left(\sum_{t=1}^{T} x_t \right) / \delta \tag{19}$$

Theorem 2: (Maximal *r*)

The maximum value for *r* is

$$max(x_t | t) = 1, ..., T \tag{20}$$

Proof: With $r = max(x_t | t) = 1, ..., T$ the demand of tokens can be fulfilled every period by the rate and the bucket depth can be set to zero.

Theorem 3: (Minimal *B*)

The minimal value of *B* is

$$B_{min} = \max_{1 \leq t \leq T} (x_t - r). \tag{21}$$

Proof: Knowing *r* we can find bounds for B_{opt} too. In each period it is necessary that $r + B \geq x_t$. (21) follows directly from this.

Theorem 4: (Maximal *B* for $\delta < 1$)

The maximal value of *B* for $\delta < 1$ is

$$B_{max1} = \frac{\max_{1 \leq t \leq T} \left(rt - \sum_{i=1}^{t} x_i \right)}{1 - \delta} \tag{22}$$

Proof: At the end of each period *t,* the number of token which were produced and not used is

$$c_t = \delta B + r\,t - \sum_{i=1}^{t} x_i \tag{23}$$

The bucket depth B never has to be greater than the maximal c_t. (22) is following directly from that.

Theorem 5: (Maximal B for $\delta > 0$)

Another maximal value if B for $\delta > 0$ is given by

$$B_{max2} = \frac{\max\limits_{1 \le t \le T} \left(\sum_{i=1}^{t} x_i - r\,t \right)}{\delta} \tag{24}$$

Proof:. There must be at least one period t for which $c_t = 0$. Otherwise it would be possible to reduce B and save costs without harm. Thus it follows from (23) that $0 = \delta B - \min\limits_{1 \le t \le T} \left(\sum_{i=1}^{t} x_i - r\,t \right)$. (24) follows directly from that.

We now describe the optimal values of B for a given R:

Theorem 6: (Optimal B for $\delta > 0$).

For $\delta > 0$ and for any $r \in [0, max(x_t)]$, the optimal B is:

$$B_{opt} = max(B_{opt1}, B_{opt2}) \tag{25}$$

$$\delta B_{opt1} = \max_{1 \le v \le T} \left(\sum_{i=1}^{v} x_i - r v \right) \tag{26}$$

$$B_{opt2} = \max_{2 \le u \le v \le T} \left(\sum_{i=u}^{v} x_i - r(v - u + 1) \right) \tag{27}$$

Proof: This follows immediately from (16) and (17).

Theorem 7: For $\delta = 0$ and for any $r \in \left[\max\limits_{1 \le v \le T} \left(1/v \cdot \sum_{i=1}^{v} x_i \right), max(x_t) \right]$, the optimal B is B_{opt2} from (27).

Proof: This also immediately follows from (16) and (17).

The empirical envelope has been defined in [18] as

$$\zeta(t) = max\left\{ \sum_{\tau=s}^{s+t} x_\tau \forall s \right\}. \tag{28}$$

The optimal B for a given rate r is the Legendre transformation (see [4]) of $\zeta(t)$. The Legendre transformation is a contact transformation that describes a convex are concave curve by means of its tangent lines parametrized by their slope (here r) and their intersection the the vertical axis (here B). See also [23].

3.3.2 An algorithm for B_{opt}

We now have to strive for algorithms to calculate the expression B_{opt1} and B_{opt2} from (26) and (27) as efficiently as possible. It is clear that B_{opt1} can be easily calculated with a cost of *O(T)*. But B_{opt2} is the maximum of *T(T-1)/2* expressions; there is a way of avoiding evaluating all O(T^2) expressions. In [33] an algorithm is described that parses the input pattern only once and is of complexity *O(T)*. For this algorithm, we consider how to compute the sequence of:

$$B_t = \max_{2 \le u \le v \le t} \left(\sum_{i=u}^{v} x_i - r(v-u+1) \right), \; t = 2,...,T \tag{29}$$

recursively with respect to *t*. By Theorem 6 the B_{opt2} we are striving for is B_T. Let's use the auxiliary notion

$$D_t = \max_{2 \le u \le t} \left(\sum_{i=u}^{t} x_i - r(t-u+1) \right), \; t = 2,...,T\,. \tag{30}$$

B_t is the maximum of *t(t-1)/2* numbers and each of them corresponds to the part of flow in interval [*u*,*v*]. The value D_t is the maximum of the subset of those numbers whose interval ends at *t*. Obviously, $D_2 = x_2 - r$. If D_{t-1} is found, then D_t is either the previous maximum D_{t-1} plus the new input $x_t - r$ or simply $x_t - r$, whichever is higher.

Knowing how to compute D_t inductively, we are ready to compute B_t now. First $B_2 = D_2$. Assume B_{t-1} is found. The B_t is the maximum of the *t(t-1)/2* numbers in the interval [*u*,*v*]. If $v < t$, $B_t = B_{t-1}$. Otherwise, $v=t$ and $B_t = D_t$. In other words, when looking at the demand x_t in period *t* the bucket size either *needs* or *needs not* to be increased to account for x_t. The case that it needs to be increased is covered by building the maximum with D_t.

To summarize we have the following inductive algorithm:

```
Bucket size - algorithm
        For a traffic pattern P={x_t, t = 1, ... , T}
        and a token bucket rate r ∈ [0, max(x_t)] for δ > 0
        and r ∈ [ max_{1≤v≤T} ( 1/v × Σ_{i=1}^{v} x_i ), max(x_t) ] otherwise,
        the B_opt2 as defined in (27)equals B_T given by
        the following recursive formulas:
```

$$D_2 = B_2 = max(x_2 - r, 0) \tag{31}$$

$$D_t = max(D_{t-1} + x_t - r, x_x - r),\; t=2,...,T \tag{32}$$

$$B_t = max(B_{t-1}, D_t),\; t=2,...,T \tag{33}$$

This algorithm is not only efficient in term of CPU but in term of memory too: it needs to store only the pattern and two integers, *B* and *D*.

3.4 An efficient algorithm for the TBAP

So far we have shown how to calculate the optimal *B* for a given rate *r* with *O(T)*. Now, the optimal *r* has to be found using a more efficient algorithm than the MIP algorithm in Section 3.2. To develop this algorithm, the mathematical

properties of the curves $B_{opt} = f(r)$ and $P_{opt} = g(r)$ which describe the optimal bucket size B and costs P for a given rate r have to be analysed first.

With $B_{opt}(r)$ from Theorem 6 rsp. Theorem 7 $P_{opt}(r)$ immediately follows as

$$P_{opt}(r) = C_r \cdot r + C_B \cdot B_{opt}(r). \tag{34}$$

3.4.1 Some Mathematical Properties of $B_{opt}(r)$ and $P_{opt}(r)$

In this section we implicitly always consider

$r \in [0, max(x_t)]$ for $\delta > 0$ and

$$r \in \left[\max_{1 \le v \le T} \left(1/v \times \sum_{i=1}^{v} x_i \right), max(x_t) \right] \text{ otherwise.}$$

Theorem 8: The curve $B_{opt} = f(r)$ is piecewise linear and convex.

Proof:We know from Theorem 6 rsp. Theorem 7

$$f(r) = \max_{2 \le u \le v \le T} \left(\sum_{i=u}^{v} x_i - r(v-u+1) \right) \text{ or } f(r) = \max_{1 \le v \le T} \left(\frac{1}{\delta} \cdot \sum_{i=1}^{v} x_i - rv \right).$$

Let´s write $f_{u,v}(r) = \sum_{i=u}^{v} x_i - r(v-u+1)$ for $2 \le u \le v \le T$ and $f_{1,v}(r) = \frac{1}{\delta} \sum_{i=1}^{v} x_i - rv$ for $1 \le v \le T$.

The function $f(r)$ can also be defined as the maximum of these $T(T+1)/2$ linear functions. $f(r)$ is therefore also piecewise linear. Depending on the value of r, $f(r)$ will be defined by on one or another of these $T(T+1)/2$ functions. Actually, it is clear that some of the functions $f_{u,v}(r)$ will never be used: there are only T different slopes for these functions (the values between *1* and *T*) and it is clear that if for a given r a linear function h takes a smaller value than another linear function l with the same slope, then $h(r) \le l(r)$ for all r. Therefore, $f(r)$ consists of the maximum of up to T linear functions $f_k(r) = A_k - k \times r$ with $k = 1,...,T$ and $A_k \ge 0$ (see Figure 8). It is clear that for increasing r function $f(r)$ is defined of the f_k with a higher slope (smaller k). In other words, the higher the value of r, the smaller the value of k. As the slope of $f(r)$ increases with r, $f(r)$ is convex.

Theorem 9: The curve $P_{opt} = g(r)$ is piecewise linear and convex

Proof:$g(r) = C_r \cdot r + C_B \cdot B_{opt}(r)$ with C_r and $C_B > 0$. $g(r)$ is the sum of two piecewise linear and convex functions[2]. g is therefore piecewise linear and convex(see Figure 9).

[2] The function h(r) = a*r is convex and concave at the same time.

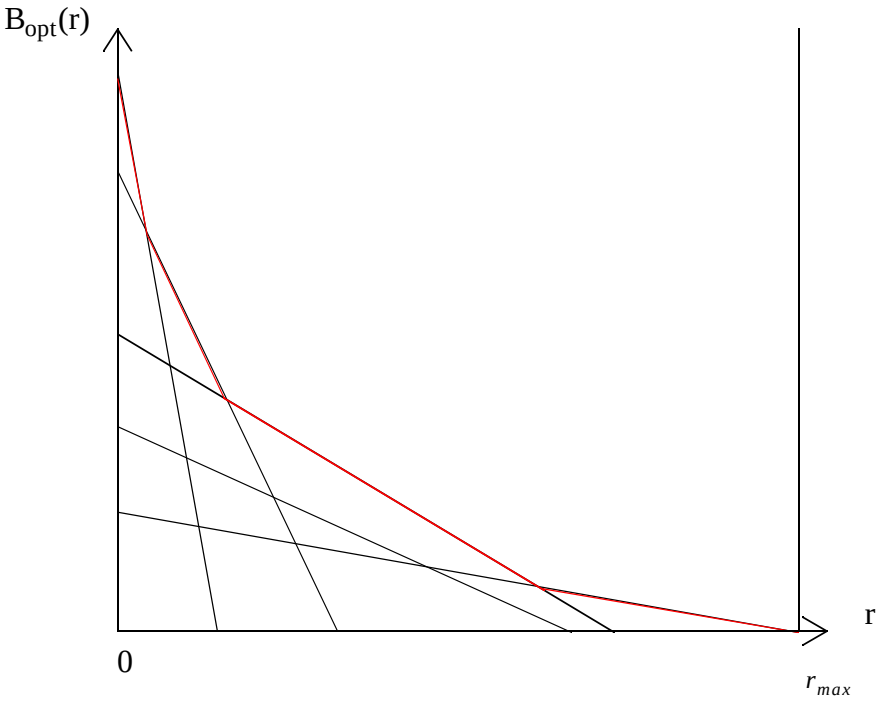


Figure 8: $B_{opt}(r)$ ($\delta > 0$)

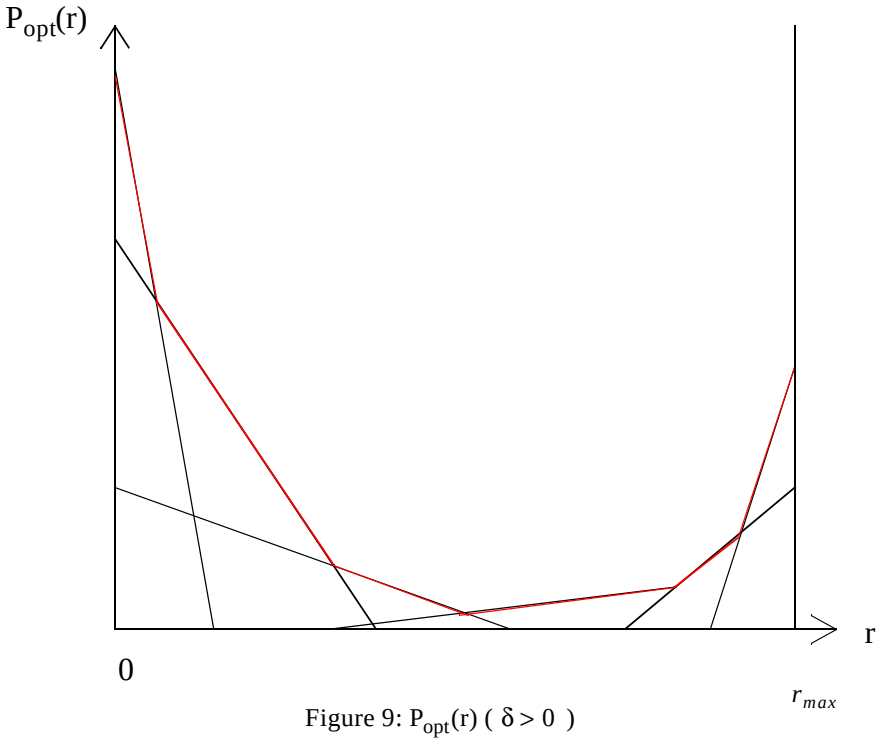


Figure 9: $P_{opt}(r)$ ($\delta > 0$)

Theorem 10: If three distinct points of a convex curve are on the same line, then the curve is a straight line between these three points.

Proof: Let´s take three points A (x_A , y_A), B (x_B , y_B) and C (x_C , y_C) of a convex curve with $x_A < x_B < x_C$.

Because of the convexity of the curve, we know that its slope can not decrease between A and C. So, if A, B and C are on the same line it means that the slope of the curve didn't change, and then, the curve must be a straight line between A and C.

3.4.2 Search for the optimal *r*

Solving the TBAP means finding the minimum $P_{opt}(r)$, defined on the interval $[r_{deb}, max(x_t)]$, with $r_{deb} = 0$ if $\delta > 0$ and $r_{deb} = \max_{1 \le v \le T}\left(1/v \times \sum_{i=1}^{v} x_i\right)$ otherwise, knowing that this function is convex and piecewise linear. The algorithm of Section 3.3.2 is used to find $P_{opt}(r)$ with a cost of $O(T)$ for a given r. The algorithm presented now exploits the properties of the curve $P_{opt}(r)$ so that we only need to calculate the coordinate of a very small number of r to find the minimum.

Because of the convexity of P_{opt}(r), the function $P_{opt}(r)$ will first decrease, then reach its minimum P^* in r^* (maybe maintain this value in an interval [r^*_1 , r^*_2]) and will then increase again (see Table 1).

$[\, r_{deb} \,,\, r^*_1 \,]$	$[\, r^*_1 \,,\, r^*_2 \,]$	$[\, r^*_2 \,,\, max(x_t \mid \forall t) \,]$
decrease	constant at P*	increase

Table 1: Variations of P_{opt}

Note that the cases $r^*_1 = r_{deb}$ (the minimum is reached for $r^* = r_{deb}$), $r^*_1 = r^*_2$ (the minimum is reached for a unique $r = r^*$) and $r^*_2 = max(x_t)$ (the minimum is reached for $r^* = max(x_t)$) are possible and have to be considered. Let´s take any two values r_1 and r_2 of $[r_{deb}, max(x_t)]$ with $r_1 < r_2$. If $P_{opt}(r_1) > P_{opt}(r_2)$ then P_{opt} decreased in a part of the interval $[r_1, r_2]$. Because of the properties of the curve P_{opt} illustrated in the Table 1, it means that $r^*_1 > r_1$, values of $r < r_1$ no longer have to be considered. With a similar argument one can argue that if $P_{opt}(r_2) > P_{opt}(r_1)$ the values $r > r_2$ can be ignored. This line of argument is valid for any interval $[r_{min}, r_{max}] \subset [r_{deb}, max(x_t)]$.

The algorithm in Figure 10 iteratively reduces the search interval for r^*: The following questions remain to be answered:

a. Which values r_1 and r_2 to choose in line 9?

b. What is the exit condition for the while loop (see line 8)?

c. What value is returned as optimal r?

Let's look at question a): The presented approach is somewhat similar to the Regula Falsi method. After an iteration either the part $[r_{min}, r_1]$ or the part $[r_2, r_{max}]$ of the interval is cut off. Choosing values for r_1 and r_2 close to each

1: Search for r*
2: if ($\delta = 0$)
3: then $r_{min} = \max_{1 \le v \le T} \left(1/v \times \sum_{i=1}^{v} x_i \right)$;
4: else $r_{min} = 0$;
5: endif
6: $r_{max} = max(x_t)$;
7: calculate $P_{min} = P_{opt}(r_{min})$;calculate $P_{max} = P_{opt}(r_{max})$;
8: while (not condition)
9: choose r_1 and r_2 with $r_{min} < r_1 < r_2 < r_{max}$
10: calculate $P_1 = P_{opt}(r_1)$;
11: calculate $P_2 = P_{opt}(r_2)$;
12: if (P1>P2)
13: then $r_{min} = r_1$; $P_{min} = P_1$;
14: else $r_{max} = r_2$; $P_{max} = P_2$;
15: endif
16: endwhile

Figure 10: Basic Algorithm to find r*

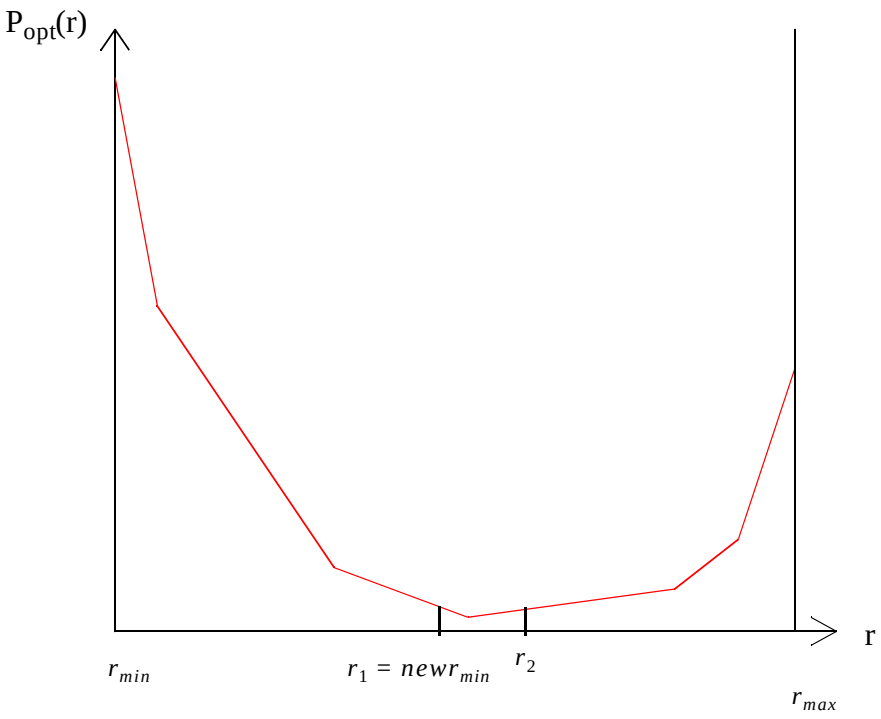


Figure 11: example of r_1 and r_2

other in the middle of the interval $[r_{min}, r_{max}]$ allows to halve the interval with each iteration. But then, the one of the two points r_1 and r_2 which was not chosen as cut off delimiter for $[r_{min}, r_{max}]$ will be very close to one end of the new interval. An example is depicted in Figure 11, $P_{opt}(r_1) > P_{opt}(r_2)$ so the new $r_{min} = r_1$. But as r_2 is now very

close to the left end of the new interval two new points in the middle of the new interval and their costs P_1 and P_2 would have to be calculated now. Because of the cost *O(T)* of these calculations we want to reduze the number of such calculations. It would be better to choose r_1 and r_2 so that on of the them can be reused in the next iteration. The new algorithm is shown in Figure 12.

```
1:        Improved Search for r*
2:              if (δ = 0)
3:                   then r_min = max_{1 ≤ v ≤ T} ( 1/v × Σ_{i=1}^{v} x_i );
4:                   else r_min = 0;
5:              endif
6:              r_max = max(x_t);
7:              r_1 = 1/3 r_max +2/3 r_min;
8:              boolean b = true;
9:              calculate P_min = P_opt( r_min );
10:             calculate P_max = P_opt( r_max );
11:             calculate P_1 = P_opt( r_1 );
12:             while (not condition)
13:                  if (b) // b is true
14:                       r_2 = (r_1 + r_max) ÷ 2;
15:                       calculate P_2 = P_opt( r_2 );
16:                       if (P_1 > P_2)
17:                                 then r_min = r_1; P_min = P_1; r_1 = r_2; P_1 = P_2;
18:                            else r_max = r_2; P_max = P_2; b = false;
19:                       endif
20:                  else // b is false
21:                       r_2 = (r_min + r_1) ÷ 2;
22:                       calculate P_2 = P_opt( r_2 );
23:                       if (P_1 > P_2)
24:                            then r_max = r_1; P_max = P_1; r_1 = r_2; P_1 = P_2;
25:                            else r_min = r_2; P_min = P_2;
26:                       endif
27:                  endif
28:             endwhile
```

Figure 12: Improved Algorithm to find r*

With this algorithm only one cost term has to be calculated each iteration. In each iteration the length of the interval will be reduced by either a half, a third, or a quarter. In the best case it will be reduced by half each iteration, in the worst case we will have a cycle a third-a quarter leading to a reduction by a half for two iterations as the following thoughts will show:

We begin with an interval of a certain length *L*. It is cut into 3 parts each of them of length L/3. Let's call this case "case A".

In the first iteration one of the two exteriors subintervals are excluded, the new intervall is of length $L_1 = 2L/3$. It is divided into two subintervals each of them with a length of $L_1/2$. A new point is created resulting again in 3 subintervalls, one of them with a length of $L_1/2$, the two others with a length of $L_1/4$. That's "case B". During the iteration, one of the subintervals will be excluded. If it is the big one (of length $L_1/2$) this leads to case B again with a smaller interval. But if it is one of the smaller ones an intervall of length $L_2 = 3L_1/4$ divided into a subinterval of length $L_2/3$ and a subinterval of length $2L_2/3$. The big subinterval will be halfed which leads to case A

Summarizing case A leads to B after a reduction of the length of the interall of 1/3 and B leads either to A after a reduction of 1/4 or to B again with a reduction of 1/2. In the best case we stay in B and the intervall is reduced by half each iteration.

The worst case are the cycles A-B with a reduction of 1/3 each second iteration and 1/4 each other iteration. Thus after 2 iterations the length of the intervall is 3/4*2/3=1/2 it's initial length.

This also shows that the algorithm is better than just doing a binary search with two new points each iteration, which would lead to halfing the interval for each two new points calculated which equals the outcome of the worst case only of our algorithm.

3.4.3 Break case of the algorithm

Now that we know how to iteratively reduce our search interval we have to decide when to end the while loop in line 8 of Figure 10 rsp. line 12 of Figure 12.

The curve $P_{opt}(r)$ is composed of at most T line segments. If there are three or more segments left to search through, one can be sure that after a limited number of iterations of the algorithm in Figure 12 one of them will disappear from the search interval. But as soon as there are only two segments left in the search interval, the following iterations do not necessarily cut off one more of the segments. Because in the worst case the interval is reduced by half in two iterations, we know that after a number of order $O(log(T))$ of iterations only one or two segments will be left in the interval. But how can one detect that only one or two segments are left in the search interval? And how can one find the optimal point of the curve with that knowledge?

- The special case where there is only one segment left is easy: one just has to test wether the points (r_{min}, P_{min}), (r_1, P_1), (r_2, P_2) and (r_{max}, P_{max}) are on a line. It is clear that if that is not the case there is more than one segment left. And because of the Theorem 9 this necessary condition is a sufficient one, too. So, if all the points are aligned the optimal point is the minimum of (r_{min}, P_{min}) and (r_{max}, P_{max}). This test has to be done each iteration.
- But in the general case we will have to stop when there are still two segments. If the test above indicates there is more than one segment left we have to test wether there are two segments. To do so we calculate the intersection of the $((r_{min}, P_{min}), (r_1, P_1))$ with the line $((r_2, P_2), (r_{max}, P_{max}))$ and check if it belongs to the curve $g(r)$.

  If three of these points are on the same line, then the intersection will be either the point (r_1, P_1) or the point (r_2, P_2).

Let´s assume it is the point (r_1, P_1). In this case we know (r_1, P_1), (r_2, P_2) and (r_{max}, P_{max}) are on a same segment (see Figure 13). We also know that the optimal point can not be right from (r_2, P_2). The next iteration

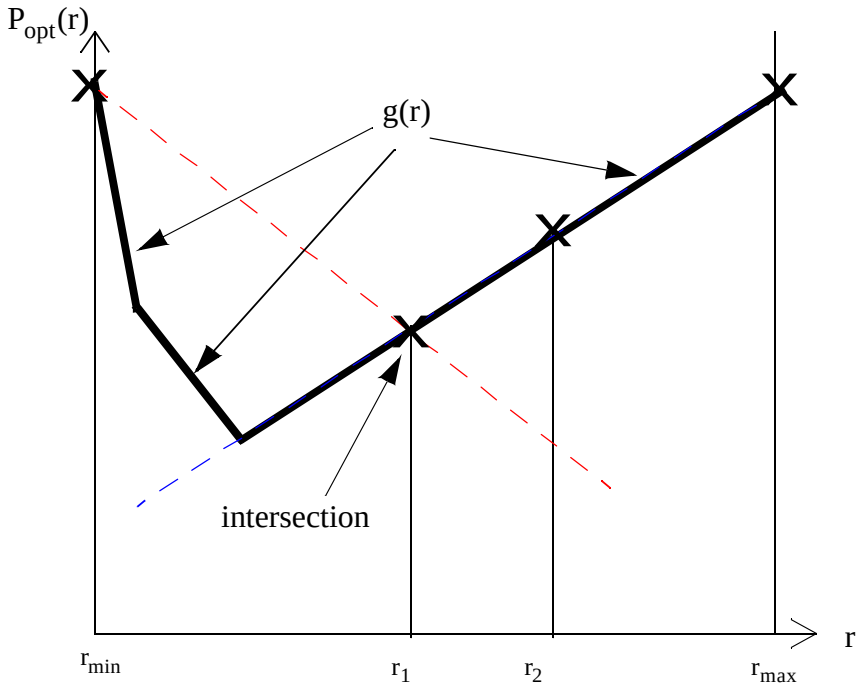


Figure 13: Three points on the same segment

will therefore be with (r_{max}, P_{max}) replaced by (r_2, P_2).

- Now look at the case where there are only two segments left. Line $((r_{min}, P_{min}), (r_1, P_1))$ is the first and the line $((r_2, P_2), (r_{max}, P_{max}))$ to the second one. The intersection of the two lines will also belong to the curve and the optimal point will be either the intersection point or (r_{min}, P_{min}) or (r_{max}, P_{max}).
- If, on the other hand, there are three segments or more the intersection (r_i, P_i) of the two lines the intersection can not belong to the curve. If it did then the three points (r_{min}, P_{min}), (r_1, P_1) and (r_i, P_i) would be on the same line, and the three points (r_i, P_i), (r_2, P_2), (r_{max}, P_{max}) would be on an other same line. That means there is only one segment between r_{min}, r_1 and r_i and only one between r_i and r_{max}. So there would be only two segments in our interval, what is incompatible with our assumption.

So we have a simple way to recognize the two break cases of our algorithm: when the four points are aligned, we have only one segment in our interval, and when the intersection of the two lines $((r_{min}, P_{min}), (r_1, P_1))$ and $((r_2, P_2), (r_{max}, P_{max}))$ belongs to the curve we have two. Now the algorithm for the TBAP is complete.

3.4.4 The Complete Algorithm for the TBAP.

```
1: void optimal_rate()
2:         if (δ = 0)
```

```
3:              then r_min = max_{1 ≤ v ≤ T} ( 1/v × Σ_{i=1}^{v} x_i );
4:              else r_min = 0;
5:         endif
6:         r_max = max(x_t);
7:         r_1 = 1/3 r_max +2/3 r_min ;
8:         boolean b = true;
9:         boolean condition = false;
10:        /* b indicates if the next point we calculate will be after or
11:           before the one we kept from the last iteration.
12:           condition indicates if the optimal point has been found.*/
13:        float r_i ; float P_i; float r_opt , float P_opt;
14:        /* r_i and P_i will be used to store the coordinates of the intersection.
15:           In r_opt and P_opt the coordinates of the optimal point are stored */
16:        calculate P_min = P_opt( r_min );
17:        calculate P_max = P_opt( r_max );
18:        calculate P_1 = P_opt( r_1 )
19:        while (not condition)
20:             if (b)then
21:                  r_2 = (r_1 + r_max) ÷ 2 ;
22:                  calculate P_2 = P_opt( r_2 );
23:                  condition = test (r_min , P_min, r_1 , P_1, r_2 , P_2, r_max , P_max)
24:                  /* the function test (see below) returns true
25:                     if there is only one or two segments in the
26:                     interval.*/
27:                  if (condition) then
28:                       if ((P_min < P_i) and (P_min < P_max))then
29:                            r_opt = r_min ;
30:                            P_opt = P_min;
31:                       else
32:                            if ((P_max < P_i) and (P_max < P_min))
33:                                 r_opt = r_max ;
34:                                 P_opt = P_max;
35:                            else
36:                                 r_opt = r_i ;
37:                                 P_opt = P_i;
38:                            endif
39:                       endif
40:                       /* if there are only one or two segments in the interval,
41:                          the optimal point is either one of the ends of the interval,
42:                          or the intersection of the two segments. If not we have to
43:                          reduce the length of our interval and to make a new
44:                          iteration*/
45:                  else // if (not condition)
46:                       if (P_1>P_2)
47:                            then r_min = r_1; Pmin = P_1; r_1 = r_2 ; P_1 = P_2;
48:                            else r_max = r_2; Pmax = P_2; b = false;
49:                       endif
50:                  endif // end if (condition) ... else
```

```
51:             else // if (not b)
52:                  r_2 = (r_min + r_1) ÷ 2 ;
53:                  calculate P_2 = P_opt( r_2 );
54:                  condition = test (r_min , P_min, r_2 , P_2, r_1 , P_1, r_max , P_max)
55:                  if (condition){
56:                       if ((P_min < P_i) and (P_min < P_max))
57:                            r_opt = r_min ;
58:                            P_opt = P_min;
59:                       else
60:                            if ((P_max < P_i) and (P_max < P_min))
61:                                 r_opt = r_max ;
62:                                 P_opt = P_max;
63:                            else
64:                                 r_opt = r_i ;
65:                                 P_opt = P_i;
66:                            endif
67:                       endif
68:                  else // if (not condition)
69:                       if (P_1>P_2)
70:                            then r_max = r_1; P_max = P_1; r_1 = r_2 ; P_1 = P_2;
71:                            else r_min = r_2; P_min = P_2;
72:                       endif
73:                  endif // end if (condition) ... else
74:             endif // end if (b) ... else
75:        end while
76:end void optimal_rate

77:void test( r_A , P_A, r_B , P_B, r_C , P_C, r_D , P_D){
78:*/ the function test returns true if there are only one or two linear segments
79:   between r_A and r_B .*/
80:        if (( r_A , P_A), ( r_B , P_B), ( r_C , P_C), ( r_D , P_D) aligned)
81:             /* here we test if they belong to ONE segment */
82:             condition = true;
83:             r_i = r_A ;
84:             P_i = P_A;
85:        else
86:             /* if it is more than one segment we modify the position of one of
87:                the points until there are no three of them on the same segment*/
88:             if (( r_A , P_A), ( r_B , P_B), ( r_C , P_C), aligned)
89:                  condition = false;
90:             else
91:                  if (( r_B , P_B), ( r_C , P_C), ( r_D , P_D), aligned)
92:                       condition = false;
93:                  else
94:                       /* and then we test if the intersection of the two lines
95:                          defined by our four points belong to the curve.
96:                          If yes it means that there are only two segments in
97:                          our interval*/
98:                       calculate (r_i, P_i) =
99:                            intersection (( r_A , P_A), ( r_B , P_B))
```

```
100:                              with (( r_C , P_C), ( r_D ,P_D))
101:                         if (P_i == calculate P_opt(r_i ))
102:                              condition = true
103:                         else
104:                              condition = false
105:                         endif
106:                   endif
107:              endif
108:         endif
109:end void test
```

3.4.5 Performance Evaluation

As explained before the *average* number of iterations will be in *O(log(T))*. The calculation of the minimal costs for a given *r* has a CPU cost of *O(T)*. In each iteration we need to calculate one new point, plus second one, if there is more than one segment in the interval to test if the intersection belongs to the curve. Sometimes we will need some more calculations if three of our points are in a line, but that is generally not the case. On average the CPU cost of this algorithm is *O(T log(T))*.

As we can see in our algorithm, we only need a very small number of variables. The most of the memory needed to store the values x_t for *T* periods. Therefore, memory costs are *O(T)*.

We implemented this algorithm in Java to compare its efficiency with the LP programming of Section 3.2. The demand were randomly generated patterns, costs were set to $C_r = 1$, $C_B = 0.1$ and the bucket starting factor to $\delta = 0.5$. The measurements were taken on a PC with a 700MHz Pentium III Processor and 256 Megabytes RAM. The average results over *n*=10 simulations are listed in Table 2. They show clearly the superiority of the iterative algorithm over the Branch-and-Bound method.

T (length of the pattern)	**CPU time (iterative algorithm)**	**CPU time (B&B method)**
50	61 ms	59 ms
100	67 ms	58 ms
200	77 ms	116 ms
500	54 ms	374ms
1000	78 ms	1,004 s
1500	75 ms	1,8 s
10000	449 ms	not enough memory
100000	2,65 s	not enough memory
1000000	16,0 s	not enough memory
2000000	37,8 s	not enough memory
4000000	52,1 s	not enough memory
6000000	1mn 17s	not enough memory

T (length of the pattern)	CPU time (iterative algorithm)	CPU time (B&B method)
7000000	not enough memory	not enough memory

Table 2: Comparison of the performances of two methods to solve a TBAP

4 The Token Bucket Reallocation Problem (TBRP)

So far we have shown how to calculate the optimal token bucket for a flow of a given length *T*. Now we look at a more complicated model by allowing reallocations of the token bucket parameters. When transmitting a longer flow the user has the option, not to request a single token bucket for all *T* periods but instead to request a series of token buckets with different parameters (r_i, B_i) and possibly different durations τ_i. Of course there has to be a mechanism that gives incentives to the user not to reallocate too often. As incentive we introduce fixed setup costs that have to be "paid" for each reallocation. We call this problem the token bucket **re**allocation problem (TBRP).

After reallocation *i* the bucket contains dB_i tokens. The number of tokens left in the old bucket B_{i-1} does not matter. Because of this property the necessary token bucket parameters and the minimal costs for a period $[t_1, t_2]$ are independent of the previous allocations.

We first discuss the cost model used for the TBRP in 4.1, then formulate the TBRP as an optimization problem in 4.2. After that we present an exact algorithm and a heuristic based on it for the TBRP in 4.3 and conclude with the presentations of other heuristics in 4.4.

4.1 A Cost Model for the TBRP

The cost model of the TBRP is more complicated than the one for the TBAP. The solution of the TBRP is a series of token buckets, each with a duration τ_i. The costs of each of those token buckets is independent of the other token buckets and depends only on the token bucket parameters (r, B) and the duration τ_i:

1. For each allocation, independent of its duration, fixed setup costs *F* have to be paid. They do not necessarily have to reflect monetary costs, this term is also used to account for technical or other fictive costs. $p_1 = F$.
2. The token rate *r* induces linear costs proportional to to height and duration: $p_2 = f(r, t_i) = \alpha \cdot r \cdot \tau_i$
3. Each unit of bucket depth can be used to increase the burst in one period by one (if the bucket is filled). The longer the communication lasts the more often the bucket can be filled and used again. The costs per bucket depth *B* must therefore dependent on the height and the duration τ_i: $p_3 = f(B, t_i) = \beta \cdot B \cdot \tau_i$
4. The *B* tokens present in the bucket at the beginning of the allocation induces costs independent of τ_i but proportional to the number of tokens $\delta \cdot B$: $p_4 = f(\delta \cdot B) = \gamma \cdot \delta \cdot B$

With the four cost coefficients *F*, α, β, and γ, the costs for one allocation are

$$P_i = f(r, B, t_i) = F + \alpha \cdot r_i \cdot \tau_i + \beta \cdot B_i \cdot \tau_i + \gamma \cdot \delta \cdot B_i \tag{35}$$

and the costs for the complete session consisting of a series of i = 1, ..., I token bucket allocations each with a duration τ_i and token bucket parameters (r_i, B_i) are

$$P = \sum_i P_i = \sum_i (F + \alpha \cdot r \cdot \tau_i + \beta \cdot B \cdot \tau_i + \gamma \cdot \delta \cdot B). \tag{36}$$

Remark: α and γ are both costs for one single token, it is reasonable to assume $\alpha \geq \gamma$ because one unit of *r* gives the same advantages as one token in the bucket in the first period and additionally can be used every period and not only once.

4.2 Formulation of the TBRP as an Optimization Problem

The TBRP can be formulated as the following optimization problem:

Variables:

- r_t rate in period $t = 1,...,T$.
- B_t bucket depth in period $t = 1,...,T$.
- y_t number of tokens in the bucket at the beginning of the period $t = 1,...,T$.
- z_t binary variable, set to 1 if the token bucket parameters (r_t, B_t) were changed at the beginning of the period $t = 1,...,T$ and 0 otherwise. This variable is necessary to account for the fixed setup costs F.

Parameters:

- x_t tokens used in period $t = 1,...,T$ to send data.
- α cost coefficient for the rate r_t.
- β cost coefficient for the bucket depth B_t.
- γ cost coefficient for each token in the bucket at the beginning of a new allocation period.
- F fixed setup costs per reallocation.
- δ bucket starting factor ($\delta \in [0, 1]$).
- M big enough constant to resemble infinity numerically, e.g. $M = \sum_{t=1}^{T} x_t$)

$$\textit{Minimize} \quad F\sum_{t=1}^{T} z_t + \alpha\sum_{t=1}^{T} r_t + \beta\sum_{t=1}^{T} B_t + \gamma\delta\sum_{t=1}^{T} (z_t B_t) \tag{37}$$

subject to

$$r_t + y_t \geq x_t \qquad \textit{for all } t = 1,...,T \tag{38}$$

$$y_t \leq (1 - z_t)B_t + z_t\delta B_t \qquad \textit{for all } t = 1,...,T \tag{39}$$

$$y_t \leq (1 - z_t)(y_{t-1} + r_{t-1} - x_{t-1}) + z_t M \qquad \textit{for all } t = 2,...,T \tag{40}$$

$$B_t - B_{t-1} \leq Mz_t \qquad \textit{for all } t = 1,...,T \tag{41}$$

$$B_{t-1} - B_t \leq Mz_t \qquad \textit{for all } t = 1,...,T \tag{42}$$

$$r_t - r_{t-1} \leq Mz_t \qquad \textit{for all } t = 1,...,T \tag{43}$$

$$r_{t-1} - r_t \leq Mz_t \qquad \textit{for all } t = 1,...,T \tag{44}$$

$$r_t, b_t, y_t \geq 0 \qquad \textit{for all } t = 1,...,T \tag{45}$$

$$z_t \in \{0, 1\} \qquad \textit{for all } t = 1,...,T \tag{46}$$

The target function (37) minimizes the total costs consisting of the fixed setup costs that are accounted for whenever z_t is 1, the costs per period for the rate r_t and bucket depth B_t plus the costs for the tokens that are in the bucket when a new allocation starts (marked by $z_t = 1$).

Constraint (38) expresses that the rate and number of tokens in the bucket must be high enough to satisfy the demand of tokens x_t each period. (39) and (40) account for the tokens left in the bucket. Two situations have to be distinguished. If z_t in period t is 1, then a reallocation is performed and (39) and (40) simply constrain the tokens in the bucket to the start value expressed with δ. Otherwise ($1-z_t$) is 1 and the number of tokens left in the bucket are restricted by

the bucket size in (39) and by how many were used in the previous period in (40). (41) to (44) force z_t to 1 if one of the parameters r_t oder B_t changes. (45) and (46) are the non-negativity rsp. binary conditions for the variables.

Tthis optimization problem a quadratic optimization problem and thus generally very hard to solve exactly with standard techniques [13]. We therefore directly strive for algorithms that exploit the special structure of the problem.

4.3 Dynamic Programming (DP)

4.3.1 Exact Version

We can solve the TBAP (see Section 3) between each couple of periods u,v with $1 \le u \le v \le T$ and store the TB parameters (r, B) and the costs of these $T(T+1)/2$ problems. On this data we can run the following dynamic programming algorithm that calculates the "shortest path" from 1 to T resulting in the optimal series of token buckets.

```
1: void DP()
2:         double Prov;
3:         double Costs [ ] = new double [ T ];
4:         int Next [ ] = new int [ T ];
5:         for( int u = T ; u > 0 ; u--)
6:              Costs [ u ] = getsolutionTBAP(u, u + T - 1) + F;
7:              Next [ u ] = T ;
8:              for( int v = u ; v < T - 1; v++ ){
9:                   Prov = getsolutionTBAP(u, v) + F + Costs [ v + 1 ];
10:                  if ( Prov < Costs [ u ]) then
11:                       Costs [ u ] = Prov;
12:                       Next [ u ] = v + 1;
13:                  end if
14:             end for v
15:        end for u
16:end void
17:
18:void getsolutionTBAP(u,v)
19:// returns the costs of the optimal TBAP solution between u and v
20:// see Section 3.4.4
```

Result: *Costs[0]* contains the total cost of the optimal solution while the array *Next* stores the hops (series of token buckets) of that solution.

The CPU costs for finding the best path is $O(T^2)$ but we have to solve $T(T+1)/2$ TBAPs too, each one with an average cost of $O(T \log(T))$, so that the total CPU costs of this algorithm are $O(T^3 \log(T))$. The memory costs are $O(T)$.

4.3.2 Heuristic Version

The algorithm above is an exact algorithm. We now describe the DP heuristic which is the heuristic verision of the DP algorithm above.:

Before we solve the TBAP between periods u and v, we have a look at the previous solution found for u and $v-1$.

- If the rate $r(u, v-1)$, the token Bucket size $B(u, v-1)$ and the numbers of token remaining at the end of the period $v-1$ $y_{v-1}(u, v-1)$ are big enough to satisfy the demand of period v, then we will take $r(u,v)=r(u, v-1)$ and $B(u,v)=B(u,v-1)$ and calculate the corresponding cost $p(u,v) = p(u, v-1)$ and token remaining in the bucket.

  $y(u, v) = \min (B(u,v), y_{v-1}(u, v-1) + r(u, v) - x_v)$.

The parameters we calculate this way are not always optimal but in general they will be close to the optimal parameters.

- Only if the previous rate and left tokens are not sufficient we solve the TBAP for (*u, v*).

The resulting algorithm looks as follows:

```
1: void DP_Heuristic()
2:         double r = 0;
3:         double B = 0;
4:         double y = 0;
5:         double Costs[ ] = new double [ T + 1 ] ;
6:         int Next [ ] = new int [ T ];
7:         Costs [ T ] = 0.0;
8:         double Prov = 0;
9:         Costs [ T - 1 ] = getsolutionTBAP(´T, T) + F;
10:        Next [ T - 1 ] = T;
11:        for( int u = T - 2; u > - 1 ; u--)
12:             Prov = getsolutionTBAP(u, u) + F;
13:             Costs [ u ] = Prov + Costs [ u + 1 ];
14:             Next [ u ] = u + 1;
15:             r = getrate(u, u);
16:             B = getsbucketsize(u, u);
17:             y = getremainingtoken(u, u);
18:             for( int v = u + 1; v < T; v++)
19:                  if( y + r < x_v)
20:                       Prov = getsolutionTBAP(u, v) + F;
21:                       r = getrate(u, v);
22:                       B = getsbucketsize(u, v);
23:                       y = getremainingtoken(u, v);
24:                  else
25:                       Prov = Prov + alpha * r + beta * B;
26:                       y = y + r - x_v - 1;
27:                       if ( y > B )
28:                            y = B;
29:                       endif
30:                  endif
31:                  if (Prov + Costs [ v + 1 ] < Costs [ u ])
32:                       Costs [ u ] = Prov + Costs [ v + 1 ];
33:                       Next [ u ] = v + 1;
34:                  endif
35:             next v
36:        next u
37:end void
```

With this DP heuristic solving the TBAP with a CPU cost of *O(T log(T))* can be exchanged with the simple operation with a CPU cost of *O(1)*. However we cannot predict how often this exchange can be done. In the worst case we have the same complexity as above [$O(T^3 \log(T))$]. In the best case the complexity is $O(T^2)$. The memory costs are again *O(T)*.

4.4 Merge, Split and Combined Heuristics

We now look for some heuristics to solve the TBRP. The following heuristics are inspired by similar heuristics that have proven themselfs for related problems in [8] and [9].

4.4.1 Merge and Split Heuristics

The merge heuristic (MH) starts with a separate allocation for each single period and then tries to merge two successive allocations into one if the saved fixes costs are less than the additional costs for *r* and *B.* The split heuristic (SH) starts with one allocation over all periods and then tries to split existing allocations into two if the additional fixed costs for the new allocation are less than the saved costs for *r* and *B*.

To describe these algorithms we use a structure *group* with the attributes *start, end, cost, last* and *next* and the functions *merge()* and *split()*. A *group* represents the set of the periods from *start* to *end*, *cost* is the cost to pay for exaclty one allocation *start* and *end*, *last* indicates if the current group is the last one (then *end* = *T*) and otherwise *next* points to the next group. The method *merge()* takes two groups and a cost as arguments and merges the two groups and attributes the new cost to the new group. The method *split()* takes a group, an integer and two costs as arguments and splits the group into two at the integer given as argument and with the two given costs.

The merge heuristic looks as follows:

```
1: void MH()
2:         Group G = new Group;
3:         Group Gr = G;
4:         double p;
5:         for(int t = 0; t < T - 1; t += 1 )
6:              Gr.start = t + 1;
7:              Gr.end = t + 1;
8:              Gr.last = false;
9:              Gr.cost = getsolutionTBAP(t+1, t+1) + F;
10:             Gr.next = new Group();
11:             Gr = Gr.next;
12:        next t
13:        Gr.start = T ;
14:        Gr.end = T ;
15:        Gr.last = true;
16:        Gr.cost = getsolutionTBAP(T, T) + F;
17:        boolean test = true;
18:        while ( test )
19:             test = false;
20:             Gr = G;
21:             while ( Gr.last == false )
22:                  p = getsolutionTBAP(Gr.start, Gr.next.end)) + F;
23:                  if ( p < Gr.cost + Gr.next.cost )then
24:                       Gr.merge( p );
25:                       test = true;
26:                  else
27:                       Gr = Gr.next;
28:                  endif
29:             endwhile Gr.last
30:        endwhile test
31:end void
```

The split heuristic can be written as:

```
1: void SH()
2:         Group G = new Group();
3:         Group Gr = G;
4:         double p1;
5:         double p2;
6:         G.start = 1;
7:         G.end = T;
8:         G.last = true;
9:         G.cost = getsolutionTBAP(1, T) + F;
```

```
10:        boolean test = true;
11:        while  ( test )
12:             test = false;
13:             Gr = G;
14:             for ( int t = Gr.start; t < Gr.end; t += 1 )
15:                  p1 = getsolutionTBAP(Gr.start, t) + F;
16:                  p2 = getsolutionTBAP(t+1, Gr.end) + F;
17:                  if ( p1 + p2 < Gr.cost ) then
18:                       Gr.split( t, p1, p2 );
19:                       test = true;
20:                       Gr = Gr.next;
21:                  endif
22:             next t
23:             while ( Gr.last == false )
24:                  Gr = Gr.next;
25:                  for ( int t = Gr.start; t < Gr.end; t += 1 )
26:                       p1 = getsolutionTBAP(Gr.deb, t)+F;
27:                       p2 = getsolutionTBAP(t+1, Gr.end) + F;
28:                       if ( p1 + p2 < Gr.cost ) then
29:                            Gr.split( t, p1, p2 );
30:                            test = true;
31:                            Gr = Gr.next;
32:                       endif
33:                  next t
34:             endwhile last
35:        endwhile test
36:end void
```

The memory costs of these algorithms are of *O(T)*. The CPU costs are complicated to evaluate. Each time that all the costs for a set of groups covering all the period from 1 to *T* are calculated, the costs are *O(T log(T))*. That is done each iteration. For the split heuristic, each time that a group is split into two, in the next iteration the split heuristic is applied separately to each group. It will be stopped in the worst case when the groups are only one period long. So the worst case number of iterations is *O(log(T))*. Using a similar argument the number of iterations in the merge heuristic is *O(log(T))*, too. This results in the worst case CPU costs of $O(T \log^2(T))$ for both heuristics.

4.4.2 Combining the Merge and Split Heuristics

A way to find a better solution is to combine both heuristics MH and SH by applying them iteratively until the solution doesn't improve any more:

```
1: void combi()
2:         Group G = new Group;
3:         double p1;
4:         double p2;
5:         G.start = 1;
6:         G.end = T;
7:         G.last = true;
8:         G.cost = getsolutionTBAP(1, T) + F;
9:         double pr = 0;
10:        double pr1 = -1;
11:        while ( pr1 < pr){
12:             pr = G.SH();
13:             pr1 = G.MH();
14:        endwhile
15:end void
```

The result cannot be worse than the best result of merge and split.

5 Numerical Simulation

We now describe the results from several numerical simulations of TBRPs.

5.1 Basic Simulation

5.1.1 Setup

The simulation was done with Java as programming language on a PC with a 700MHz Pentium III Processor and a 256 MB RAM.

The basic simulation uses the video traces patterns of [27]. These 21 traces are from MPEG versions of different types of video sequences (movies, cartoons, TV, sport...). For the first experiment we aggregate each trace so that one period represents one group of pictures (12 frames, 0.5 seconds). 2000 periods would therefore be equal to little more than 15 minutes of a movie. The average bit rate of the movies is 0.536 Mbps, the average peak rate of the movies 3.54 Mbps.

The cost coefficients are $\alpha = \gamma = 1$; $\beta = 0.1$ and $F = 10^5$, the bucket starting factor is set to $d = 0.5$.

The TBRP was solved for different values of T ranging between 10 and 2000 by each algorithm of Section 4 and by the TBAP algorithm of Section 3.4. We measured the CPU time, the numbers of allocations and the relative difference of between the calculated cost and the optimal cost (given by the dynamic programming algorithm of Section 4.3.1) defined by $\Delta = \frac{P - P_{opt}}{P_{opt}}$. The tables and figures in this chapter are based on the mean values over the results from the 21 traces.

5.1.2 Results

The results of the basic simulation are given in the following tables and figures.

T	TBAP	SH	MH	Combi	DPH	DP
10	35,2 ms	110 ms	134 ms	134 ms	17,6 ms	127 ms
50	64,3 ms	201 ms	250 ms	266 ms	78,6 ms	605 ms
100	40 ms	267 ms	640 ms	460 ms	190 ms	1960 ms
200	52,9 ms	445 ms	1520 ms	904 ms	580 ms	9330 ms
500	41 ms	1040 ms	4270 ms	2650 ms	2950 ms	99000 ms
1000	65,5 ms	2730 ms	9080 ms	6340 ms	14400 ms	11 mn 12 s
2000	103 ms	9890 ms	18000 ms	16800 ms	49900 ms	1h 23mn

Table 3: Computation time

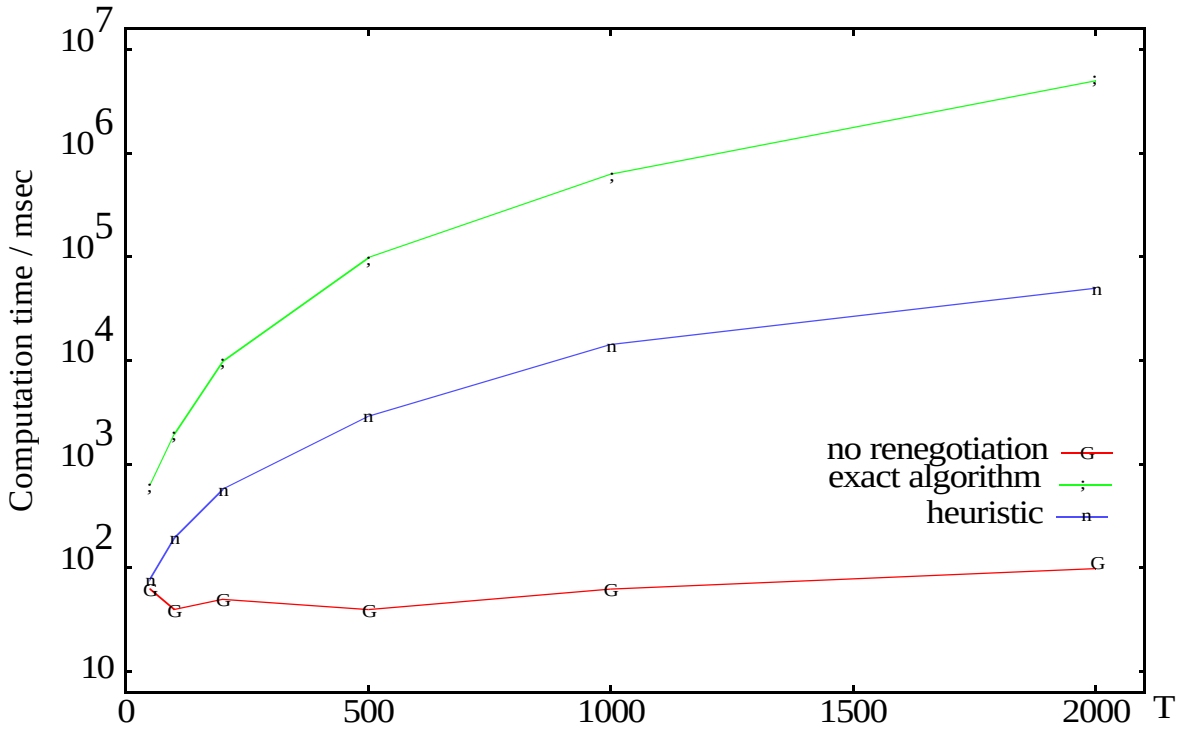


Figure 14: Computation time (logarithmic)

T	TBAP	SH	MH	Combi	DPH	DP
10	+9%	+2,42%	+2,13%	+0,483%	+0,337%	-
50	+26,5%	+8,54%	+6,11%	+1,93%	+0,204%	-
100	+39,3%	+12,3%	+6,85%	+1,4%	+0,215%	-
200	+53,7%	+16,2%	+7,99%	+2,07%	+0,2%	-
500	+82,6%	+23,9%	+7,18%	+2,32%	+0,206%	-
1000	+113%	+27,2%	+7,88%	+2,02%	+0,23%	-
2000	+131%	+30,6%	+7,5%	+1,92%	+0,22%	-

Table 4: Relative difference in costs to the optimal solution

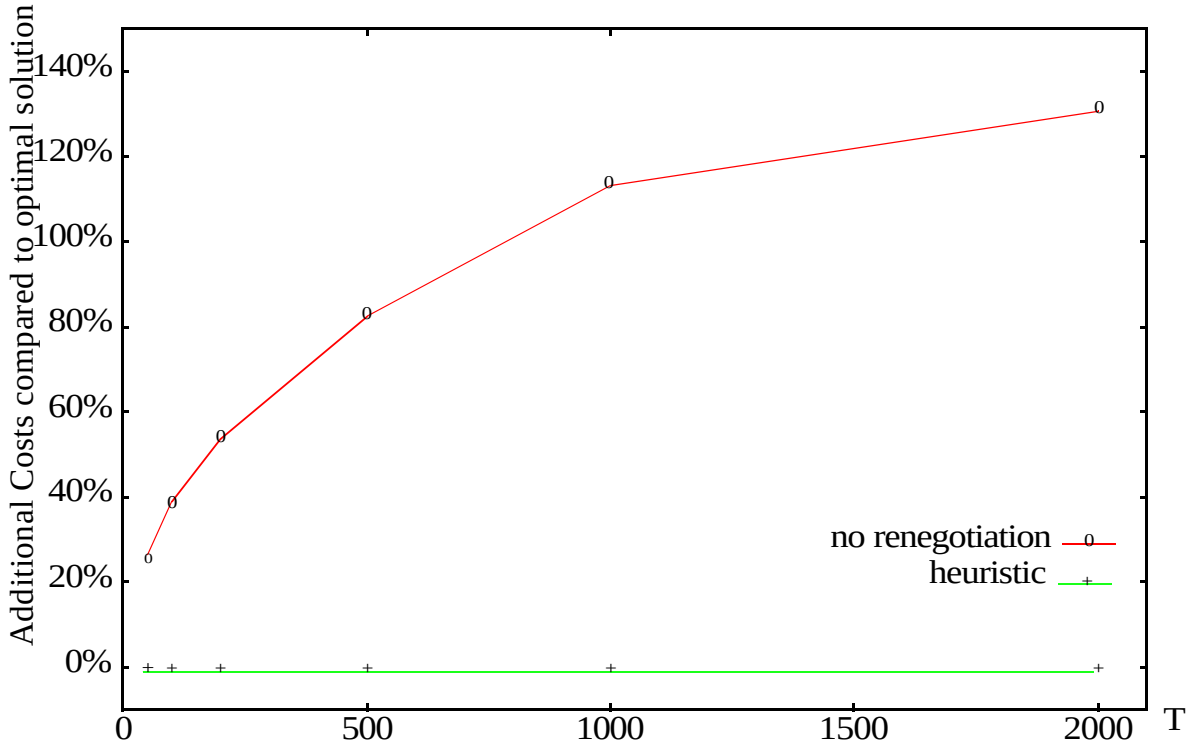


Figure 15: Relative difference to optimal costs

T	TBAP	SH	MH	Combi	DPH	DP
10	1	2,71	1,52	2,05	2,05	2,1
50	1	14,3	3,81	5,52	6,43	6,57
100	1	40,4	6,19	10	11,4	11,5
200	1	97,5	12	19,2	22	22,3
500	1	340	30	49,3	54,4	54,7
1000	1	776	60,1	102	109	109
2000	1	1681	118	199	219	220

Table 5: Number of allocations

5.1.3 Discussion

By looking at the computation times one first notices that the DP algorithm takes by far the longest time to solve as can be expected as it has the highest computational complexity. The DPH heuristic is much faster than the DP algorithm. This indicates that in practice it can avoid solving a lot of TBAPs because it can extend the previous token bucket by just one period in most of the cases. Generally the merge heuristic MH takes more time to solve than the split heuristic SH. The combination of both always takes longer than SH alone because SH is the first step of the combined heuristic followed by other (smaller) steps of MH and then SH again. Usually the combined heuristic is still faster than MH alone. This shows that it was a wise idea to start with SH in the combined heuristic. As the combined heuristic cannot by definition yield worse results than MH alone, MH is dominated by the combined heuristic. If one compares the computation time of SH and the combined heuristic with that of DPH the latter is generally faster for smaller but slower for larger problems. This is an indication that it has an average computational complexity higher than $O(T)$.

The fastest way is to solve the TBRP as a TBAP and just execute the TBAP algorithm and use the single allocation it returns. But for T=10 periods one can see that the DPH heuristic is faster than the TBAP algorithm which is surprising at the first look because it internally also uses the TBAP algorithm. But it uses it for subproblems with less than 10 periods and extends the solution of those with fast and easy operations to 10 periods, that is why it can on average be faster than the TBAP alone for problems with very few periods.

More important than the execution time is generally the quality of the results measured by the relative difference of the costs compared to the optimal costs that DP returns[3]. The simple and fast algorithm TBAP yields much higher costs than all the other algorithms. The difference increases with the number of periods which is obvious as the potential benefit of being able to reallocate increases with T. The bad results also shows that it generally makes sense to reallocate and to look at the token bucket reallocation problem TBRP as it can very significantly reduce costs by a factor of 2 and more. The DPH algorithm is extremely close to the optimal solution, resulting only in less than 0.25% higher costs. SH yields bad results while MH does far better. The combination of both is significantly better than both of the single algorithms and only roughly 2% away from the optimum.

Looking at the average number of allocations in Table 5 it is obvious that TBAP only uses one allocation for all periods. The optimal solution reallocates roughly speaking every 9 to 10 periods. MH results into less allocations than SH is obvious by the nature of these algorithms. In the simulation both numbers are far away from the optimal number of allocations. The combination of both results in a number much closer to the optimal number and lies in between the numbers of MH and SH. The difference between DPH and DP is extremely small as one would expect from its good results regarding the cost.

5.1.4 Conclusion

The merge and especially the split heuristic alone cannot be recommended, the combined heuristic performed quite well. Lower costs can be gained with the DPH heuristic at the cost of a little more execution time for higher T. DPH is extremely close to the optimal results yielded by DP but performs a lot faster and scales better; for most situations DPH should therefore be the preferred algorithm.

[3] The entry for the exact algorithm DP in Table4 is obviously zero.

5.2 Increasing the Fixed Setup Costs F

5.2.1 Setup

In the basic simulation reallocations occurred relatively often every 9 to 10 periods. We now increase the fixed setup costs by a factor of 10 to $F=10^6$.

Because of the long time it takes to simulate all 21 movie traces we concentrate on a single trace (cartoon "Asterix"). The results from the basic simulation for this movie are also shown for comparison.

5.2.2 Results

The results are displayed in the following tables, the results from the basic simulation are marked "old F" while the new results are marked "new F".

T	SH		MH		Combination		DPH		DP	
	old F	new F	old F	new F	old F	new F	old F	new F	old F	new F
10	0,11 s	0,16 s	0,22 s	0,33 s	0,17 s	0,11 s	0 s	0 s	0,11 s	0,16 s
50	0,22 s	0,22 s	0,33 s	0,38 s	0,33 s	0,22 s	0,11 s	0,11 s	0,55 s	0,55 s
100	0,33 s	0,27 s	0,82 s	0,38 s	0,55 s	0,38 s	0,16 s	0,11 s	1,98 s	1,98 s
200	0,28 s	0,72 s	1,32 s	1,65 s	0,99 s	0,94 s	0,33 s	0,33 s	9,72 s	9,94 s
500	0,72 s	2,25 s	2,53 s	4,56 s	2,31 s	3,4 s	4,62 s	4,61 s	1mn 45s	1 mn 46s
1000	2,37 s	8,51 s	9,39 s	14,8 s	5 s	8,95 s	10,6 s	10,6 s	10mn 31s	10mn 19s
2000	9,23s	9,89 s	17,5 s	47,2 s	15 s	19,2 s	45,4 s	45,5 s	1h 48 mn	1h 19 mn

Table 6: Computation time (Asterix)

T	TBAP		SH		MH		Combination		DPH		DP	
	old F	new F	old F	new F	old F	new F	old F	new F	old F	new F	old F	new F
10	5,63%	0%	1,27%	0%	0%	0%	1,27%	0%	0%	0%	0%	0%
50	23,3%	2,53%	7,65%	2,17%	1,78%	2,53%	1,02%	2,17%	0%	0%	0%	0%
100	32,1%	9,97%	7,61%	5,84%	2,14%	9,97%	1,01%	2,93%	0,0271%	0%	0%	0%
200	42,2%	18,4%	12,8%	10,8%	4,44%	18,4%	9,68%	3,75%	0,182 %	0%	0%	0%
500	65,9%	37,3%	27,3%	33,1%	4,51%	21,3%	2,06%	2,96%	1,28%	0,15%	0%	0%
1000	151%	109%	22,9%	58,3%	3,98%	63,5%	1,68%	4,2%	0,182 %	0,14%	0%	0%
2000	151%	109%	24,1%	57,1%	3,7%	87%	1,7%	3,36%	0,281 %	0,599 %	0%	0%

Table 7: Difference in costs to the optimal solution (Asterix)

T	TBAP		SH		MH		Combination		DPH		DP	
	old F	new F	old F	new F	old F	new F	old F	new F	old F	new F	old F	new F
10	1	1	2	1	2	1	2	1	2	1	2	1
50	1	1	20	2	6	1	7	2	9	2	9	2
100	1	1	38	4	10	1	14	3	17	4	18	4
200	1	1	35	10	16	1	15	5	25	6	29	6
500	1	1	446	75	36	2	56	13	60	18	63	18
1000	1	1	815	238	78	3	126	30	124	38	125	37
2000	1	1	1703	466	164	2	254	54	260	70	264	70

Table 8: Number of allocations (Asterix)

5.2.3 Discussion

Comparison of the Asterix movie with the basic simulation If one compares the old results of the Asterix trace here with the results for the average over all video traces in Section 5.1 one notices that the Asterix trace results in a slightly above average number of reallocations but generally all other observations made in Section 5.1 hold true for this special trace.

Comparison of the different niveaus of F Now we compare the results for the new niveau of fixed costs with the old one. As expected the number of reallocations drops significantly. The average duration of an allocation increases from roughly 9 periods to 27 to 28 periods. The results of MH are very extreme, the number of reallocations goes down close to 1, resulting in an extreme performance loss regarding costs. Also SH loses performance. The combination of MH and SH is still far better than both heuristics alone but also lost some performance compared to the old fixed cost niveau.

On the other hand the TBAP algorithm gains performance because the more a reallocation costs the less weighs his drawback that he never reallocates. The already good results of DPH get still slightly better in all but one case but generally the niveau of costs remain extremely close to the optimum. The CPU time of DP and DPH does not change significantly. That is not surprising because both algorithms spend most of their time creating a table for the time ranges (u,v) with $1 \le u \le v \le T$ and this calculation is not influenced by the difference in fixed costs.

5.2.4 Conclusions

An increase by 10 in fixed costs resulted in a decrease of roughly factor 3 in the number of allocations, so the fixed costs are a way to control the number of reallocations. The results of this simulation show again that the pure heuristics MH and SH should be avoided, the basic results from Section 5.1 remain untouched.

5.3 Frames instead of Group of Pictures

5.3.1 Setup

So far one period represented one group of pictures (GOP). Now we look at a smaller period size where one period corresponds to one frame (1/24 s). We compare the results for the movie "Asterix", first when one period corresponds to the duration of one GOP, then when it corresponds to the duration of one frame. The costs per period are kept the same in both simulations.

5.3.2 Results

The results are as follows:

T=	**TBAP**		**SH**		**MH**		**Combination**		**DPH**	
	GOP	**frame**	**GOP**	**frame**	**GOP**	**frame**	**GOP**	**frame**	**GOP**	**frame**
10GOP = 120frames	5,63%	16,7%	1,27%	5,07%	0%	16,7%	1,27%	0,61%	0%	0%
50GOP = 600frames	23,3%	45,5%	7,65%	24,8%	1,78%	42,1%	1,02%	3,7%	0%	0%
100GOP = 1200frames	32,1%	57,1%	7,61%	26,7%	2,14%	54,6%	1,01%	2,87%	0,0271%	0%
200GOP = 2400frames	42,2%	71,9%	12,8%	30,5%	4,44%	48,2%	9,68%	3,85%	0,182%	0%

Table 9: Cost difference to the optimal solution (GOP/frames)

T=	**TBAP**	**SH**	**MH**	**Combina-tion**	**DPH**	**DP**
10GOP = 120frames	+47,5%	+38,5%	+55,8%	+32,6%	+33,5%	+33,5%
50GOP = 600frames	+52,5%	+49,7%	+80,4%	+32,6%	+29,2%	+29,2%
100GOP = 1200frames	+53,1%	+51,6%	+94,9%	+31,1%	+28,7%	+28,8%
200GOP = 2400frames	+59,5%	+52,7%	+87,3%	+24,9%	+31,7%	+32%

Table 10: Increase in absolute costs when using a period of one frame

T=	SH		MH		Combination		DPH		DP	
	GOP	**frame**	**GOP**	**frame**	**GOP**	**frame**	**GOP**	**frame**	**GOP**	**frame**
10GOP = 120frames	2	7	2	1	2	5	2	5	2	5
50GOP = 600frames	20	61	6	2	7	12	9	17	9	17
100GOP = 1200frames	38	118	10	14	2	22	17	28	18	28
200GOP = 2400frames	35	265	16	3	15	48	25	49	29	49

Table 11: Number of allocations (GOP/frames)

5.3.3 Discussion

As can be seen from Table 10 the general cost niveau increases for the optimal solution by roughly 30% if one period represents one frame instead of one GOP. The explanation is complex:

- First of all the number of periods increases by a factor of 4 and as the costs per period remain the same one expects a strong increase in total costs.
- But on the other hand the bandwidth demand per period decreases by a factor of 4 on average. One could expect somewhat similar decrease in the parameters *r* and *b* and in the costs they cause.
- Both effects do not cancel each other out, the first effect is stronger than the second, mainly because there is a higher variety in the demands if looking at single frames instead of GOPs so that *B* cannot be reduced by a factor of 4. Also the fixed costs do not decrease and as there are more periods and reallocations this increases total costs, too.

By looking at the relative cost difference DPH again is extremely close to the optimum. MH and SH performe worse on frames than on GOPs. This can be explained by the fact that the number of periods *T* is higher if performed on frames and this also resulted in worse results for these heuristics in the previous simulations for these heuristics.
The number of allocations increases if looking at frames too which is not suprising and caused by the increasing number of periods.

5.3.4 Conclusions

The finer the granularity the higher the costs if the costs per period are kept constant.

5.4 Simulations with Short- and Long-Range Dependent Traffic

5.4.1 Setup

Here we test our model with three patterns random generated using the *fft_fgn* traffic generator [25, 31]. The three patterns are partially normal distributed. The first pattern is a pure normal distributed pattern (Hurst parameter H=0.5), the second a normal distributed pattern with a low autocorrelation of the values (Hurst parameter H=0.7) and the third a normal distributed pattern with a strong autocorrelation of the values (Hurst parameter H=0.9).

5.4.2 Results

The results are the following ones: .. .. ..

T=	TBAP	SH	MH	Combina-tion	DPH	DP
10	110ms	220 ms	220 ms	220 ms	< 50 ms	160 ms
50	< 50 ms	0,380s	0,330 s	0,380 s	110 ms	0,880 s
100	50 ms	1,04 s	0,71 s	0,88 s	0,16 s	3,08 s
200	100 ms	1,75 s	1,32 s	1,49 s	0,39	14,4 s
500	60 ms	4,45s	7,36 s	3,52 s	1,65 s	2 m 19 s
1000	60 ms	17,6 s	20,3 s	10,2 s	12,4 s	17 m 34s
2000	60 ms	17,7 s	46,9 s	29,0 s	25,3 s	1h 46 m

Table 12: Computation Time (H=0.5)

T=	TBAP	SH	MH	Combina-tion	DPH	DP
10	+0%	+0%	+0%	+0%	+0%	+0%
50	+23,3%	+6,54%	+3,75%	+2,45%	+20,4%	+0%
100	+26,2%	+8,92%	+3,64%	+2,34%	+10,1%	+0%
200	+28,1%	+6,69%	+3,49%	+1,65%	+5,46%	+0%
500	+32,5%	+6,96%	+3,81%	+2,06%	+4,01%	+0%
1000	+31,2%	+6,93%	+3,54%	+1,90%	+4,05%	+0%
2000	+33,5%	+8,29%	+3,80%	+3,05%	+5,77%	+0%

Table 13: Relative cost difference to the optimal solution (H=0.5)

T=	TBAP	SH	MH	Combina-tion	DPH	DP
10	1	1	1	1	1	1
50	1	14	7	6	7	7
100	1	43	16	14	18	17
200	1	75	28	33	31	34
500	1	174	66	74	80	81
1000	1	384	144	179	168	169
2000	1	719	289	296	349	352

Table 14: Number of allocations (H=0.5)

T=	TBAP	SH	MH	Combina-tion	DPH	DP
10	< 50ms	0,170 s	0,210 s	0,270 s	60 ms	0,170 s
50	< 50ms	0,440 s	0,330 s	0,380 s	50 ms	0,880 s
100	< 50ms	1,10 s	0,820 s	0,880 s	0,160 s	3,07 s
200	110 ms	1,43 s	1,70 s	1,70 s	0,380 s	14,1 s
500	< 50ms	6,65 s	7,19 s	4,28 s	1,75 s	2 m 13 s
1000	50 ms	18,8 s	21,2 s	8,78 s	7,64 s	13 m 47 s
2000	0,110 s	13,7 s	18,0 s	27,0 s	15,9 s	1h 36 m

Table 15: Computation Time (H=0.7)

T=	TBAP	SH	MH	Combina-tion	DPH	DP
10	+0%	+0%	+0%	+0%	+0%	+0%
50	+28,5%	+6,39%	+2,11%	+2,92%	+1,71%	+0%
100	+34,6%	+8,39%	+2,42%	+0,795%	+0,215%	+0%
200	+40,0%	+6,44%	+2,59%	+1,02%	+0,108%	+0%
500	+51,6%	+8,26%	+3,75%	+1,67%	+0,998%	+0%
1000	+48,4%	+10,6%	+3,35%	+2,10%	+2,81%	+0%
2000	+53,2%	+11,7%	+3,28%	+2,45%	+2,69%	+0%

Table 16: Relative Cost difference with the optimal solution (H=0.7)

T=	TBAP	SH	MH	Combina-tion	DPH	DP
10	1	1	1	1	1	1
50	1	13	5	6	6	6
100	1	51	13	16	17	17
200	1	79	25	33	35	34
500	1	211	60	87	86	86
1000	1	506	143	178	174	172
2000	1	1055	274	341	342	355

Table 17: Number of allocations (H=0.7)

T=	TBAP	SH	MH	Combina-tion	DPH	DP
10	60ms	0,170s	0,220 s	0,170 s	< 50ms	0,160 s
50	< 50ms	0,440s	0,330 s	0,440 s	0,110 s	0,880 s
100	50 ms	0,660 s	1,38 s	0,660 s	0,170 s	3,24 s
200	0,110 s	1,32 s	1,71 s	1,21 s	0,330	14,8 s
500	< 50ms	7,19 s	7,36 s	4,12 s	2,14 s	2 m 24 s
1000	50 ms	24,2 s	8,4	9,01 s	9,94 s	13 m 45s
2000	0,110 s	12,6 s	29,5	21,2 s	18,9 s	1h 45m

Table 18: Computational Costs (H=0.9)

T=	TBAP	SH	MH	Combina-tion	DPH	DP
10	+0%	+0%	+0%	+0%	+0%	+0%
50	+26,6%	+6,19%	+0,99%	+2,71%	+0%	+0%
100	+34,0%	+8,91%	+1,33%	+1,37%	+0%	+0%
200	+44,5%	+6,66%	+2,07%	+0,83%	+0,46%	+0%
500	+66,4%	+10,1%	+3,22%	+1,17%	+0,414%	+0%
1000	+59,6%	+13,8%	+2,88%	+1,47%	+0,434%	+0%
2000	+73,0%	+13,6%	+3,28%	+1,55%	+0,927%	+0%

Table 19: Relative cost difference to the optimal solution (H=0.9)

T=	TBAP	SH	MH	Combina-tion	DPH	DP
10	1	1	1	1	1	1
50	1	14	5	6	6	6
100	1	51	13	13	15	15
200	1	87	24	32	28	28
500	1	252	58	78	74	76
1000	1	685	128	162	155	155
2000	1	1242	242	317	305	308

Table 20: Number of allocations (H=0.9)

By looking at the results for the TBAP algorithm one sees that with increasing long range dependency in the network traffic the results get worse (at least for higher *T*). This is plausible, the more long range dependency there is the more renegotiations are worth. The number of reallocations of the optimal solution does on the other hand not significantly change.

If there is no autocorrelation in the traffic the combination of MH and SH yields better results than the DPH heuristic. This can be explained easily: DPH extends the token bucket of a previous calculation by one period t+1 if the bucket

is big enough. This extension is the better the more the traffic of t+1 depends on the values t, t-1, ... that is the higher the autocorrelation is.

5.4.3 Conclusions

Apart from the fact that for low autocorrelation the DPH heuristic looses performance the same observations as made before hold true.

5.5 Summary

The results showed that renegotiations can significantly reduce the costs, most of the time by a factor of 2 and more. It was therefore sensible to look at the reallocation problem.

For all the kind of patterns we have studied, the algorithms MH and SH are not very interesting. Their combination performes far better than the two alone but is dominated in nearly all cases by DPH. DPH has a very good trade-off between computation time and the quality of the solution. As it is very close to the optimum for realistic network traffic it can be recommended even instead of the exact DP algorithm.

6 Multiregression Analysis

In this part of our work we will use the methods of multiregression analysis (as described, for example in [7]) to analyze how good r and B can be predicted by certain statistical parameters of the flow. Our aim is to find coefficients $a^r{}_i$ and $a^B{}_i$ for a TBRP of length T so that r and B can be predicted by

$$\hat{r} = a_r^0 + \sum_{i=1}^{n} a_r^i \times z_i \text{ and} \tag{47}$$

$$\hat{B} = a_B^0 + \sum_{i=1}^{n} a_B^i \times z_i \tag{48}$$

where the z_i are some easily measured statistical parameters of the flow x_t. Of course the prediction will not generally be the exact or optimal solution. After explaining the general idea of using a special regression model to improve the results for the TBAP and TBRP we use the regression to first analyse how r and B are correlated with flow parameters such as the mean data rate and the variance in Section 6.2. Then we try to exploit the results in order to further improve the dynamic programming heuristic of Section 4.3 in Section 6.3.

6.1 Regression Model

The classical methods of multiregresssional analysis (see for example [7]) are used to calculate the coefficients a_r, b_r, c_r, d_r, a_B, b_B, c_B and d_B with

$$\hat{r} = a_r + b_r \times mean + c_r \times max + d_r \times deviation \tag{49}$$

and

$$\hat{B} = a_B + b_B \times mean + c_B \times max + d_B \times deviation \tag{50}$$

for each combination of T and δ where *mean*, *max* and *deviation* stand for:

$$mean = \frac{1}{T} \sum_{t=1}^{T} x_t \tag{51}$$

$$max = \max_{t = 1, \ldots, T} (x_t) \tag{52}$$

$$deviation = \sqrt{\frac{1}{T} \sum_{t=1}^{T} (x_t - mean)^2} \tag{53}$$

6.2 Correlation

6.2.1 Simulation

The following measurement is based on the same data as in the basic simulation of Section 5.1, T is varied from 10 to 3000 and the starting factor is set to δ =0, 0.5, and 1.0. For each T we cut the movie patterns into as many subpatterns

of length *T* as possible and use all these subpatterns as basis for the regression analysis. The results are listed in the following tables:

delta	T=10	T=20	T=50	T=100	T=200	T=500	T=1000	T=2000	T=3000
0	2005,3	2705,1	321,59	2907,8	14843	-2959,2	40451	-10787	-41633
1/2	43,743	2296,8	-2207,2	14236	11495	-8077,6	-12690	-115377	-9766,4
1	-2070,6	728,85	-3302,1	-796,86	-1996,3	-46153	-34555	-127336	-24734

Table 21: experimental values for a_r

delta	T=10	T=20	T=50	T=100	T=200	T=500	T=1000	T=2000	T=3000
0	0,68135	0,72731	1,024	1,0006	0,91444	0,57137	-0,2225	0,9373	-0,7040
1/2	1,3125	0,57911	-0,0210	0,11843	0,24659	0,51563	0,40442	0,54487	1,6688
1	1,3688	1,4084	0,68380	-0,4448	-0,1539	0,37397	0,51873	0,68860	1,6738

Table 22: experimental values for b_r

delta	T=10	T=20	T=50	T=100	T=200	T=500	T=1000	T=2000	T=3000
0	0,29846	0,56352	-0,0384	-0,0550	-0,0738	-0,1016	-0,1461	-0,3371	-0,0997
1/2	-0,3227	0,37041	0,99023	0,88351	0,92025	0,96603	0,97862	0,86961	1,1178
1	-0,3366	-0,4108	0,33523	1,1821	1,0793	1,068	0,95554	0,89301	1,1695

Table 23: experimental values for c_r

delta	T=10	T=20	T=50	T=100	T=200	T=500	T=1000	T=2000	T=3000
0	0,19521	0,17516	1,1089	1,4821	1,9461	3,7301	6,1734	6,1097	9,1897
1/2	0,61212	0,79879	-0,4195	-0,3671	-0,7841	-1,4113	-1,4698	0,54155	-5,7802
1	0,30009	1,0036	1,0054	-0,5690	-0,8768	-1,5929	-1,4131	0,11014	-6,1760

Table 24: experimental values for d_r

delta	T=10	T=20	T=50	T=100	T=200	T=500	T=1000	T=2000	T=3000
0	-492,83	-3103,0	20790	19332	-46608	1341,0	-93927	-321551	197324
1/2	5786,7	-6528,3	6257,5	-54289	-24388	62967	131246	430031	71520
1	10316	6567,0	9714,3	-10701	15047	135209	205533	466591	118671

Table 25: experimental values for a_B

delta	T=10	T=20	T=50	T=100	T=200	T=500	T=1000	T=2000	T=3000
0	-0,0223	-0,0812	-2,0540	-2,7798	-2,6716	-1,9257	-0,0432	-4,0579	-0,5140
1/2	-1,7320	-1,0951	0,26136	-0,3342	-0,9219	-1,5513	-0,8812	-1,6107	-3,9833
1	-1,0415	-2,2570	-1,7549	1,7220	0,41456	-1,0094	-1,2321	-2,0501	-4,0884

Table 26: experimental values for b_B

delta	T=10	T=20	T=50	T=100	T=200	T=500	T=1000	T=2000	T=3000
0	0,00645	-0,1361	1,5880	2,2155	2,5846	3,1305	2,8961	2,1497	2,9193
1/2	1,6449	1,1659	-0,2258	0,29591	0,11408	0,10036	0,00614	0,28600	-0,2651
1	0,81488	1,9603	1,4506	-0,7753	-0,3810	-0,2102	0,12030	0,21445	-0,3938

Table 27: experimental values for c_B

delta	T=10	T=20	T=50	T=100	T=200	T=500	T=1000	T=2000	T=3000
0	2,0706	4,6380	1,9027	0,43410	-1,6420	-8,5439	-12,355	3,3592	-18,049
1/2	1,7293	-0,7316	2,2254	1,6402	3,6836	3,9798	3,4389	-1,2902	13,502
1	3,4141	1,9443	-0,7603	2,3697	3,7454	4,6745	2,8056	0,02863	14,594

Table 28: experimental values for d_B

A measure for the quality of the prediction is the correlaction coefficient that is given for the results in the following tables:

delta	T=10	T=20	T=50	T=100	T=200	T=500	T=1000	T=2000	T=3000
0	0,94756	0,92037	0,91911	0,92328	0,89899	0,89527	0,84308	0,78156	0,82952
1/2	0,97828	0,91912	0,92422	0,90147	0,89898	0,91214	0,87199	0,94078	0,90073
1	0,98253	0,97300	0,88151	0,92835	0,89437	0,90893	0,86747	0,94332	0,90741

Table 29: correlation coefficient cor_r

delta	T=10	T=20	T=50	T=100	T=200	T=500	T=1000	T=2000	T=3000
0	0,65544	0,67107	0,66047	0,68510	0,69796	0,63840	0,57567	0,57451	0,45938
1/2	0,79216	0,39838	0,28419	0,32541	0,32102	0,29501	0,17331	0,17687	0,52026
1	0,87786	0,83094	0,42669	0,35193	0,28009	0,24746	0,16375	0,20747	0,53203

Table 30: correlation coefficient cor_B

6.2.2 Interpretation

The correlation coefficient for *r* is quite good, especially for smaller *T*. That means that *r* can be predicted quite well with the regression model. Looking at the different coefficients a_r, b_r, c_r, and d_r one notices that they vary extremely for different *T* and *d*. The only regularity for a_r seems to be that it decreases for increasing *d*. This could be because for increasing *d* there are more buckets available in the first periods that might allow for a lower *r*. This effect however, should vanish for higher *T* but it does not. There is no explanation for this behaviour.

For b_r one would expect a value of around 1. This is often the case but not always, also negative values appear sometimes. One would not expect a strong influence of the maximum on the rate. This can be seen by c_r which is usually significantly below 1. The standard deviation has also often only a small impact on *r*, but there are exceptions where d_r has quite a high value. But this only occurs for higher *T* where the standard deviation usually has only a low value.

Summarizing, *r* can be predicted quite well with the multiregression model but the regression parameters depend heavily on the *T*, *d* combination and there are not many regularities explaining the values of the regression parameters.

The correlation coefficient for *B* is very bad, even for only 10 periods of *T* it is too bad to estimate a good *B.* With such a low correlation it is also unnecessary to look at the regression coefficients.

In order to better evaluate the regression results we tested the coefficients with the traces of Section 5.4. The following table lists the relative difference between the exact results for *r* and *B* obtained by the algorithm from Section 3.4 and the results from the multiregresseion analysis:

T	**10**	**100**	**1000**	**3000**
r (H=0.5)	8,86%	26,1%	23,9%	36,9%
r (H=0.7)	9,57%	21,3%	18,1%	43,0%
r (H=0.9)	9,39%	14,7%	18,0%	50,2%
B (H=0.5)	36,6%	118%	99,8%	103%
B (H=0.7)	36,3%	130%	107%	108%
B (H=0.9)	36,9%	118%	111%	112%

Table 31: Differences exact results/results got with statistical coefficients

As one can see the results for *r* are acceptable for smaller *T* but the results for *B* are too bad even for small *T*.

6.3 Application

So far have used the exact algorithm for the TBAP with *O(T log T)* in Section 3.4.4 for finding solutions for the TBRP in algorithms of Section 4.

Now we could use the regression to predict an *r* instead of the iterative search in Section 3.4.4 and use the algorithm from Section 3.3 to obtain the minimal *B* for this *r*. Of course we will not get an exact solution but saves time, especially if the correlation coefficients are known beforehand. This can come in handy especially for the DP algorithm of Section 4.3.1 which relies of solving a lot of TBAPs. The calculation of a TBAP using regression will be *O(T)* instead of *O(T log T)*.

On the other hand we know that the prediction of *r* is not too good and the regression coefficients cannot be predicted easily as they show no regularities. To evaluate this approach we use it for five video traces (Asterix, Atp, Bond, Dino and Fuss) and compare the costs with those of the optimal solution. For each tested *T* the lowest, average and the highest of the five cost differences are listed:

T=	**10**	**100**	**1000**	**3000**
lowest cost difference	+5,60%	+13,0%	+0,0019%	+4,57%
average cost differences	+33,8%	+39,3%	+30,5%	+39,9%
highest cost difference	+109%	+87,0%	+83,2%	+76,7%

Table 31: Relative cost difference between multiregression results and the optimum

The result is disappointing, the DPH and Combi heuristic of Section 4.3.2 and Section 4.4.2 yield far better results in shorter time.

6.4 Conclusions

Our regression model was not good enough to predict *r* and especially *B*. It is not possible to improve the results already obtained with the DPH heuristic further with multiregressionional analysis.

7 Related Work

7.1 Work Related to the TBAP

Falkner et. al. [6] use a cost function for token bucket dimensioning with mimimum costs from the perspective of a single user. They, however, assume an ATM network and on-off traffic that is not known in advance. They solve the resulting non-linear optimization problem with the Lagrangean method.

Bruno et. al. [3] study token bucket dimensioning for aggregate VoIP sources for the DiffServ Expedited Forwarding service class. Their LBAP is an aggregation of independent fluid on-off sources. They analyze the effect of token bucket parameters on the non-conformance probability. They, however, do not use a cost function or something similar and do not present an algorithm to derive the optimal pair of token bucket parameters.

Kulkarni and Gautam study in [19] the sizing of K token buckets with admission control resp. network utilization in mind. They also formulate and solve token bucket dimensioning as an explicit optimization problem but their perspective is fundamentally different to ours. While we consider minimizing the costs of one customer and expect the customer to choose his/her token bucket parameters they do not look at costs but try minimizing the sum of the rates of K customer's token buckets at the same time, taking the network provider's point of view.

Procissi et. al. analyse in [26] the influence of long range dependence in traffic on the dimensioning of token buckets. They use two cost models, one of them similar to the one used in this work, to derive an analytical model for estimating the token bucket parameters. This model explicitly takes into account the long range dependency of traffic, the $B_{opt}(r)$ curve is obtained for traffic modeled as a Fractional Brownian Motion process. As a result they can quite well estimate good token bucket parameters for Internet traffic. They, however, show no algorithm for calculating the optimal parameters for a given trace as we did.

Naudts [23] describes an efficient algorithm for calculating the optimal cell rate $r^*(\tau)$ for a given τ for the ATM generic cell rate algorithm (GCRA). As the GCRA can also be described as a continuous-state leaky bucket this is equivalent to calculating the bucket rate for a given bucket depth.

In [28] a token bucket marker is used for TCP streams and the effect of the token bucket parameters on the achieved sending rate are analysed. That paper operates with different assumptions (TCP instead of real-time traffic) and is thus complementary to this work.

7.2 Work Related to the TBRP

There are a number of works on renegotiable services [37, 36, 22, 35, 29]. Pricing for renegotiable services is considered e.g. in [22] and [29]. Grossglauer et. al. [37] propose the renegotiable constant bit rate service and show how it can be used to increase total network utilization. Knightly and Zhang [35, 36] extend this work to the renegotiable variable bit rate service (RED-VBR). They also consider sending an MPEG movie known in advance. They show that without renegotiation for certain MPEG streams only an average utilitzation of 25% can be achieved. They propose a heuristic called offline algorithm to calculate a series of token buckets for the ATM VBR service that achieve a far higher average utilization. This heuristic needs an input parameter that controls how often to segment the stream. This parameter is difficult to set. Our work presents an exact algorithm and an extremely close yet much faster heuristic instead.

Knightly and Zhang also present a second heuristic (online algorithm) that does not require the traffic to be known in advance and they propose an admission control scheme for renegotiable VBR services.

8 Summary and Outlook

This paper first explained the token bucket model and a pricing model for it. The relationship between the optimal token bucket parameters rate and bucket depth were discussed. Based on that an effective algorithm for calculating the optimal bucket depth for a given rate was presented. Using this algorithm an iterative search algorithm was presented to efficiently calculate the optimum combination of rate and bucket depth for a given linear cost model. With this the token bucket allocation problem (TBAP) which is finding the combination of rate and bucket depth with the least costs can be efficiently solved.

The next part of the paper dealt with the token bucket reallocation problem (TBRP) that results from allowing to change the token bucket parameters during the transmission. The cost minimal series of token buckets for a given media stream is wanted. We presented an exact mathematical formulation of the problem and an exact algorithm (DP) that solves the problem with $O(T^3 \log T)$. We also derived several heuristic, the best of them (DPH) closer than 0.25% to the optimal solution and is several orders faster than the exact algorithm. Therefore the TBRP can now also be efficiently solved.

Several simulations using video traces and randomly generated short and long range dependent traffic were used to evaluate the results. The simulations showed clearly that by reallocating the token bucket parameters the costs can generally be more than halved.

We also used a multiregressional analysis to study the correlation of rate and bucket depth on the average, maximum and standard deviation of the stream. The results from this could however not be used to further improve the heuristics as they were already to close to the optimum.

References

[1] D. Black, S. Blake, M. Carlson, E. Davies, Z.Wang, and W.Weiss, *An Architecture for Differentiated Services*, Informational RFC 2475, December 1998.

[2] R. Braden, D. Clark, and S. Shenker, *Integrated Services in the Internet Architecture: an Overview*, Informational RFC 1633, June 1994.

[3] R. Bruno, R. G. Garroppo, and S. Giordano, *Token Bucket Dimensioning for Aggregate VoIP Sources*, Proceedings of IEEE ATM Workshop 2000, Heidelberg, Germany, June 2000.

[4] R. Courant, D. Hilbert, *Methods of Mathematical Physics*, Vol. II, Interscience, New York, p. 32, 1965-1966.

[5] R. L. Cruz, *A calculus for network delay, Part I: Network elements in isolation,* IEEE Transactions on Information Theory 37 (1), 1991.

[6] M. Falkner, M. Devetsikiotis, and I. Lambadaris, *Minimum Cost Traffic Shaping: A User's Perspective on Connection Admission Control*, IEEE Communications Letters, pp. 257-259, Volume 3, September 1999.

[7] J. Hartung, *Statistik Lehr- und Handbuch der angewandten Statistik*, R. Oldenburg Verlag München Wien, 1993.

[8] O. Heckmann and J. Schmitt, *Multi-Period Ressource Allocation at System Edges*, Technical Report TR-KOM-2000-05, Department of Electrical Engineering & Information Technology, Darmstadt University of Technology, Germany, http://www.kom.e-technik.tu-darmstadt.de/publications/abstracts/HS00-2.html, 2000.

[9] O. Heckmann, J. Schmitt, and R. Steinmetz, *Multi-Period Resource Allocation at System Edges - Capacity Management in a Multi-Provider Multi-Service Internet*, Proceedings of IEEE Conference on Local Computer Networks (LCN 2001), November 2001.

[10] O. Heckmann and J. Schmitt, *A Taxonomy for Multi-Period Ressource Allocation Problems at System Edges*, Technical Report TR-KOM-2001-09, Department of Electrical Engineering & Information Technology, Technical University Darmstadt, Germany, http://www.kom.e-technik.tu-darmstadt.de/publications/abstracts/HS01-1.html, November 2001.

[11] J. Heinanen and R. Guerin, *A two rate three color marker*, RFC 2698, September 1999.

[12] J. Heinanen and R. Guerin, *A single rate three color marker*, RFC 2697, September 1999.

[13] F. S. Hillier and G. J. Lieberman, *Operations Research*, McGraw-Hill, 1995.

[14] J. Ibanez and K. Nichols, *Preliminary Simulation Evaluation of an Assured Service*, Internet Draft, draft-ibanez-diserv-assured-eval-00.txt, August 1998.

[15] Ilog Cplex, Mathematical Programming Optimizer, Product homepage: http://www.ilog.com/products/cplex/.

[16] S. Jamin, P.b. Danzig, S. Shenker, and L. Zhang, *A Measurement-Based Admission Control Algorithm for Integrated Services Packet Networks*, Proceedings of ACM SIGCOMM 1995.

[17] S. Keshav, *An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network*, Addison-Wesley, 1997.

[18] E.W. Knightly, D.E. Wrege, J. Liebeherr., and H. Zhang, *Fundamental limits and tradeoffs of providing deterministic guarantees to VBR video traffic*, Proceedings of SIGMETRICS 1995, pp. 98-107, 1995.

[19] G. Kulkarni and N. Gautam, *Leaky buckets : Sizing and admission control*, Proceedings of the 35th IEEE Conference on Decision and Control, http://citeseer.nj.nec.com/kulkarni96leaky.html, 1996.

[20] J.-Y. Le Boudec and Patrick Thiran, *Network Calculus. A Theory of Deterministic Queuing Systems for the Internet*, Lecture Notes in Computer Science Vol. 2050, Springer, 2001.

[21] J.-Y. Le Boudec, *Application of Network Calculus to Guaranteed Service Networks*, IEEE Transactions on Information Theory, Vol. 44, No. 3: 1087--1096, May 1998.

[22] G. Malewicz and A. Shvartsman, *An Auction-Based Flexible Pricing Scheme for Renegotiated QoS Connections and Its Evaluation*, http://citeseer.nj.nec.com/malewicz99auctionbased.html, Proc. of Seventh International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'99), 1999.

[23] J. Naudts, *Towards real-time measurement of traffic control parameters*, Computer Networks 34 (2000), P. 157-167, 2000.

[24] C. Partridge, *Manual page of TB program*, 1994.

[25] V. Paxson, *Fast approximation of self-similar network traffic*, Lawrence Berkeley Laboratory Report LBL-36750. April 1995.

[26] G. Procissi, M. Gerla, J. Kim, S. S. Lee, and M. Y. Sanadidi, *On Long Range Dependence and Token Buckets*, Proceedings of SPECTS 2001, Orlando, Fl, Jul. 2001.

[27] O. Rose, *Statistical Properties of MPEG Video Traffic and Their Impact on Traffic Modeling in ATM systems*, University of Wuerzburg. Institute of Computer Science Research Report Series, Report No. 101, http://citeseer.nj.nec.com/rose95statistical.html, Feb. 1995.

[28] S. Sahu, P. Nain, T. Towsley, C. Diot, and V. Firoiu, *On Achievable Service Differentiation with Token Bucket Marking for TCP*, Proceedings of ACM SIGMETRICS'00, Santa Clara, CA, June 2000.

[29] J. Sairamesh, D. F. Ferguson, and Y. Yemini, *An Approach to Pricing, Optimal Allocation and Quality of Service Provisioning In High-Speed Packet Networks*, Proceedings of INFOCOM'95, pp. 1111-1119, 1995.

[30] J. Schmitt, O. Heckmann, M. Karsten, and R. Steinmetz, *Decoupling Different Time Scales of Network QoS Systems*, Proceedings of the 2001 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), pages 428-435, July 2001.

[31] C. Schuler, *fft_fgn*, Research Institute for Open Communication Systems, GMD FOKUS, Hardenbergplatz 2, D-10623 Berlin, Germany.

[32] H. Su and M. Atiquzzaman, *Performance modeling of differentiated service network with a token bucket marker*, 11th IEEE Workshop on Local and Metropolitan Area Networks, Boulder CO., USA, pp. 81–82, March 18-21, 2001.

[33] P. Tang and T. Tai, *Network Traffic Characterization Using Token Bucket Model*, Proceedings of INFOCOM 1999, P. 51-62.

[34] L.Wolf, L. Delgrossi, R. Steinmetz, S. Schaller, and H. Wittig, *Issues of Reserving Resources in Advance*, Proceedings of NOSSDAV 1995, pp. 28-36, 1995.

[35] H. Zhang and E. Knightly, *RED-VBR: A Renegotiation-Based Approach to Support Delay-Sensitive VBR Video*, Multimedia Systems Journal, 5(3):164–176, May 1997.

[36] E. Knightly and H. Zhang, *Connection Admission Control for RED-VBR, a Renegotioation-Based Service*, Proceedings of IWQoS'96, Paris, France, 1996.

[37] M. Grossglauser, S. Keshav, and D. Tse. *RCBR: A simple and efficient service for multiple time-scale traffic*. Proceedings of SIGCOMM'95, pp. 219-230, Boston, MA, September 1995.

[38] S. Giordano and J.-Y. Le Boudec, *On a Class of Time Varying Shapers with Application to the Renegotiable Variable Bit Rate Service*, Technical Report SSC/1998/035, DI-EPFL, Lausanne, Switzerland, http://lrcwww.epfl.ch/~giordano/publications.html, 1998.

[39]

[40] ITU Telecommunication Standardization Sector - Study group 13, ITU-T Recommendation Q. 2963.1: Peak cell rate modification by the connection owner, 1996.

[41] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, *Resource Reservation Protocol (RSVP)*, RFC 2205, 1997.

[42]