Darmstadt University of Technology

Multi-Period Resource Allocation at System Edges (MPRASE)

Oliver Heckmann, Jens Schmitt
{heckmann;schmitt}@kom.tu-darmstadt.de

Version 1.0
October 2000

Last major update 01.02.01
Last minor update 06.03.01

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

Providing guaranteed QoS, be it statistical or deterministic, necessarily requires allocation of scarce resources. This might happen on a session or on an aggregate basis, nevertheless, it is conceivable that at least at system edges scarcity of resources, exposed in the form of non-negligible (virtual) costs, will prevail to necessitate explicit allocation of resources as opposed to pure overdimensioning. An example of this logic is constituted by the Differentiated Services (DiffServ) architecture which is largely based on explicit bilateral Service Level Agreements (SLA) between peering providers. Often such resource allocation decisions are done on a multi-period basis because resource allocation decisions at a certain point in time may depend on earlier decisions and thus it can turn out sub-optimal to look at decisions in an isolated fashion. Therefore, in this paper, we investigate a fairly large and diverse set of (network) QoS problems all of which deal with the problem of multi-period resource allocation at system edges. We devise a taxonomy for the classification of these problems and introduce a common mathematical framework under which these problems can be tackled. The ultimate goal of our work is to strive for solution techniques towards the generalized class of problems such that these are applicable in a number of scenarios which have so far not been regarded in an integrated fashion.

1 Introduction

Decentralized organization has always been at the heart of the Internet's philosophy. Actually, this is one of the key explanations for its success in a world of increasing deregulation. The decentralized organization is exhibited by the fact that the Internet consists of multiple independent providers that usually operate so-called autonomous systems which interwork with each other by peering agreements. It is interesting to note that both the number of ASes as well as the average number of ASes a given AS is peering with is increasing at a fairly high rate. The number of ASes rose from 909 in 9/95 to 4427 in 12/98 and 7563 in 10/00 [1,2]. Similarly, the average peering degree, i.e., the number of providers a certain provider has peering agreements with, rose from 2.99 in 9/95 to 4.12 in 12/98. It is also very notable that a single provider may peer with up to 1000 other providers [1].

From these observations it becomes obvious that for the efficient and dynamic provision of QoS in the Internet there is a good potential for optimization of resource allocations at provider edges. In particular, this optimization of resource allocations becomes a competitive factor for Internet providers. However, such an optimization can only be done on a multi-period basis, i.e., by making decisions based on earlier decisions about resource allocations at peering providers. In this paper, we want to deal with a general problem class called multi-period resource allocation at system edges (MPRASE) for which the peering providers have been the motivating scenario, although some other network QoS problems also fit into this problem class as we will discuss later on.

1.1 Motivating Example: Decoupling Different Time Scales of Network QoS Systems

Different time scales of providers' network QoS systems may arise due to different QoS architectures like RSVP/ IntServ (Resource reSerVation Protocol/ Integrated Services) [3], DiffServ (Differentiated Services) [4], or ATM (Asynchronous transfer Mode) [5] being used. Choosing different QoS architectures results from serving different needs, e.g., for an access and backbone provider. An access provider that has a comparatively moderate load and directly connects to end-systems may favour a fast time scale system responding immediately to the end-systems requests. A backbone provider that connects access providers respectively offers transit services is generally faced with a drastically higher load of individual transmissions, so that reaction on the time scale of individual requests is usually not possible and a slower time scale system needs to be enforced. When different time scales are in operation in heterogeneous network QoS systems, it is simply not possible to query the underlying QoS system each time an overlaid system is altering its state. Here, the system operating on a faster time scale needs to be smoothed when overlaying it onto a system that operates only on slow time scales.

A realistic configuration for access and backbone providers may be, e.g., that access providers use RSVP/IntServ to suit their customers' needs while a backbone provider uses DiffServ with a Bandwidth Broker (DiffServ/BB) to allow for some dynamics but on a slower time scale. This scenario is shown in Figure 1.

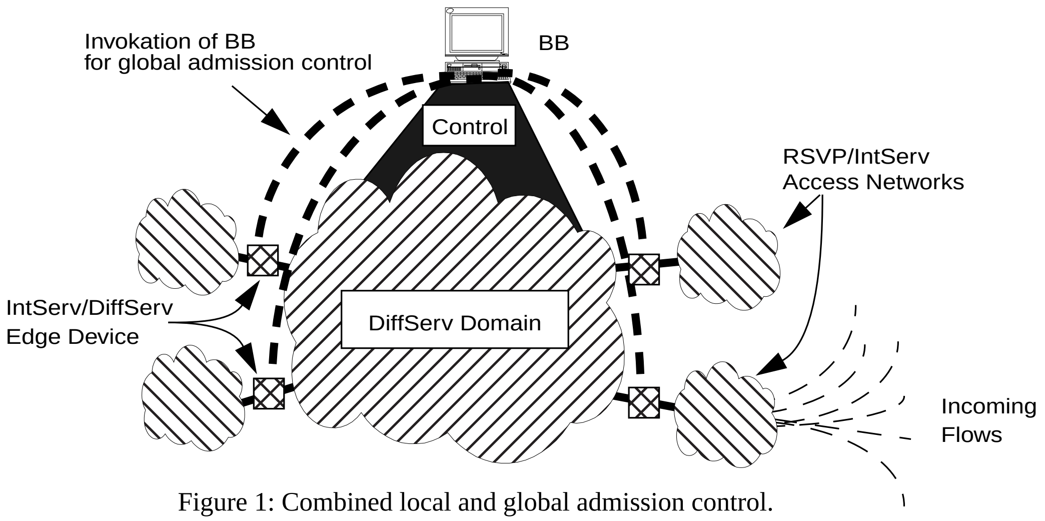


Figure 1: Combined local and global admission control.

Here it is also very obvious why a BB is generally not able to react to individual RSVP requests that are arriving at edge devices between access and backbone provider. Because if it did, the BB would need to operate at a throughput of requests that is proportional to the square of the number of access providers it serves - that is not scalable. To see this, assume each of *N* edge devices would have *M* (new or modified) RESV messages for each other edge device in a given time period and would query the BB for each of these requests. Then the BB would have to deal with $N \times (N-1) \times M$ requests in the same period. Note that the problem is not solved by spatial aggregation approaches like, e.g., [6] or [7] since for each of the *M* RESV messages the aggregate would have to be rearranged. Here a decoupling of the different time scales is necessary. The decoupling can be achieved by building "depots" of capacity which stabilize the fluctuations of the "nervous" demand curve for backbone capacity by individual requests. From another perspective, the decoupling technique can also be viewed as introducing a combined local and global admission control for the DiffServ/BB network. Global admission control is only invoked whenever local admission control at an edge device runs out of resources in its capacity depot. In such a case, local admission control on an edge device tries to obtain more resources from the global admission control represented by the BB. This scheme allows to trade off resource efficiency for a more stable and long-term capacity demand presented to the BB.

Note that the slow time scale of an underlying QoS system may not express itself in being unable to process requests for QoS at short time scales but by the fact that significant setup costs are incurred for QoS requests between different administrative domains. Such a scheme of QoS tariffing is an instance where a QoS strategy of a network provider restricts the capabilities of the employed QoS architecture. A possible reason for this may be, e.g., that the charging and accounting system is not able to deal with a large number of individual requests since this involves a lot of operational costs.

In conclusion, we have here a certain instance of the general problem class of multi-period resource allocation at system edges.

1.2 Outline

In the next section, we generalize the above presented example problem and extend it along several dimensions of a conceptual component model by which we try to capture all the different facets of the MPRASE problem structure. We show how the different characteristics of the components and their combinations lead to known network QoS problems. We thus establish the incentive to treat the generalized problem in a framework that has the potential to solve all MPRASE problems in an integrated fashion or at least establish relationships between different problem instances which may be helpful when approaching a certain problem, e.g., by making use of an existing algorithm for a certain problem as part of a solution strategy of another (more complicated) problem instance.

In Section 3, we then look at the most basic problem, the so-called single provider problem (SPP). We devise exact algorithms for its solution as well as some heuristics since it turns out to be fairly compute-intensive. Techniques for the SPP could be the basis for a solution strategy to the example problem described in Section 1.1 and in fact that is what has been proposed and evaluated in [8]. This constitutes an example where a relation between the different problem instances captured by our general model is used. In this paper, however, we try to exploit the SPP along a different dimension: the number of providers. Therefore, in Section 4, we turn to the so-called multi-provider problem (MPP). The MPP constitutes a really hard problem, however by using our techniques for the SPP we are at least able to devise heuristic approaches towards the MPP. To evaluate our techniques we carry out extensive simulations to compare the different alternatives.

In Section 5, we review related work and discuss how our approach differs from respectively extends former work. In Section 6, we summarize our findings and draw some conclusions.

2 MPRASE - Generalization & Taxonomy

In this section, we introduce a general structural model which tries to capture all the different facets of multi-period resource allocation at system edges (MPRASE) problems. This model then allows us to derive a taxonomy along its components which establishes the relations between the different problem incarnations.

2.1 Generalized Problem Structure Model

Figure 2 shows the overall structural model of the general class of MPRASE problems. Obviously, there are custom-

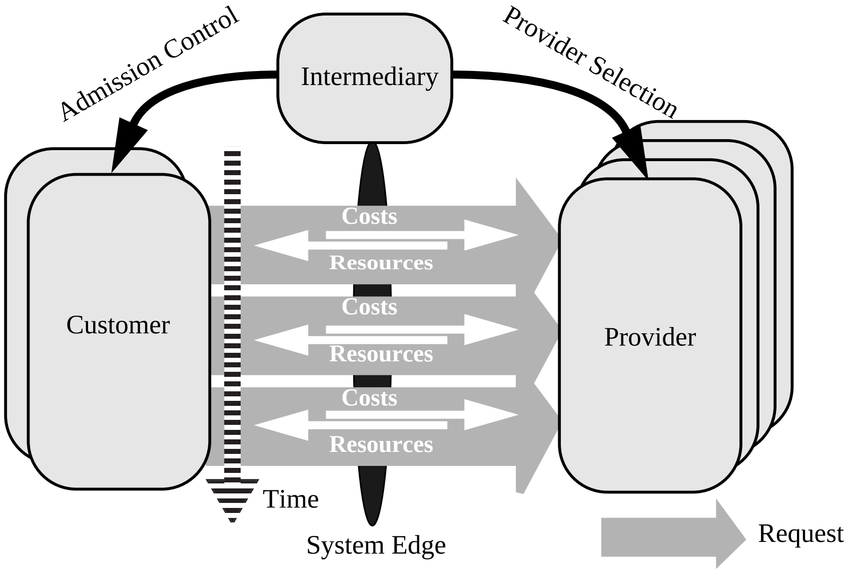


Figure 2: MPRASE problem structure.

ers which generate requests towards several providers. These two groups are separated by a system edge on which an intermediary instance is located. The intermediary tries to mediate between the two by selecting providers on the one hand and enforcing admission control of the customers on the other hand. Note that the logical separation of the intermediary instance from customer and provider does not necessarily imply that it may not belong to either customer or provider premises. The requests are originated by the customers which desire a certain amount of resources offered by the providers. Furthermore, requests incur certain costs at the providers which need to be accounted for by customers. Several requests are generated in the course of time, thereby, reacting upon the dynamics of customers' demand. Let us now look at the different components of the structural model for MPRASE.

2.1.1 Customer

The customer component of the MPRASE model captures the number of customers, i.e., if a single or multiple customers are considered, and the flexibility of the demand, i.e., whether demand may be dissatisfied or be served with a degraded quality.

2.1.2 Provider

The provider model encompasses the number of providers and whether they are modelled as having limited or unlimited capacity. While the latter is unrealistic it can be a simplifying, yet valid assumption for the case where supply exceeds demand with very high probability.

2.1.3 Resource

This component models the resources, i.e., whether they are one- or multidimensional or whether they are provided on a deterministic or statistical basis. Here, we make no particular assumption on the kind of guarantee with which resources are provided, i.e., whether they are statistically or deterministically available. Therefore, an allocation in our context does not necessarily mean an isolated, exclusive access to resources for a customer that made it.

2.1.4 Cost

The cost model seizes the cost structure for allocation requests, i.e., whether these incur certain fixed or transactional costs or whether the number of requests is just bounded and how variable costs for resource allocations are modelled, e.g., linearly or non-linearly.

2.1.5 Edge

The edge model encompasses the nature of knowledge about capacity demands at the system edges, i.e., whether deterministic or statistical knowledge about future demands is available or if total uncertainty needs to be assumed. In our work here, we focus on deterministic knowledge at system edges because methods for this case may be used as basic methods for other edge models, as is demonstrated in [8]. Furthermore, the deterministic edge model is not totally unrealistic as it applies to advance reservation scenarios as described in [9, 10] and may play an important role in DiffServ-based scenarios as described in [11].

2.1.6 Intermediary

Note that the intermediary is the component where solution techniques towards MPRASE problems are conceptually located. Therefore, this component does not capture problem characteristics but characteristics of the solution techniques, e.g., whether these are striving for the exact solution or whether they are just heuristics which could again be classified in construction and improvement techniques.

2.2 MPRASE Problem Incarnations

The above dimensions, which are largely orthogonal to each other, span a large space of MPRASE problem incarnations. Some of these - the ones considered in this paper in varying detail - are given in Table 1. The table shows

Problem	Abbrev.	Cost	Resource	Customer	Provider	Edge
Single provider problem	*SPP*	*linear fixed and variable costs*	*one-dimensional*	*single customer*	*single provider*	*deterministic*
Multi-provider problem	*MPP*	*linear fixed and variable costs*	*one-dimensional*	*single customer*	***multiple providers***	*deterministic*
capacitated MPP	*cMPP*	*linear fixed and variable costs*	*one-dimensional*	*single customer*	***multiple providers with finite capacity***	*deterministic*
Flexible demand problem with admission control	SPP_{AC}	*linear fixed and variable costs*	*one-dimensional*	***multiple reject-able customers***	*single provider*	*deterministic*
Flexible demand problem with degraded quality	SPP_{DQ}	*linear fixed and variable costs*	*one-dimensional*	***single degrad-able customer***	*single provider*	*deterministic*
Limited number of alloca-tions problem	SPP_{LAP}	***limited number of allocations, linear variable costs***	*one-dimensional*	*single customer*	*single provider*	*deterministic*
Buffered capacity alloca-tion Problem	SPP_{BCP}	*linear fixed and variable costs*	***Token Bucket***	*single customer*	*single provider*	*deterministic*
Uncertain SPP	SPP_{UE}	*linear fixed and variable costs*	*one-dimensional*	*single customer*	*single provider*	***uncertain***

Table 1: MPRASE problem incarnations.

MPRASE problem incarnations where each component is varied at least once (indicated by bold entries). This illustrates our basic goal of treating MPRASE problems in an integrated fashion by making their relations explicit and using that knowledge for solution approaches.

2.2.1 MPRASE Solution Strategies

The solution strategies can also be divided into several classes. First they can be exact or heuristic, while heuristics can be further divided into metaheuristics (like genetic algorithms or tabu search) and construction and improvement technics.

A strategy can look forward into the future in order to anticipate future development (look-ahead) or can use only input data for the actual period (myopic). If a strategy is adapting it’s behaviour in time by looking at past decisions it is called adaptiv.

Stochastic strategies can come to different solutions for the same problem instance if run twice while deterministic strategies will always come to the same solution for one specific problem instance.

2.3 Mathematical Programming Models for MPRASE

As a common framework for the general MPRASE problem class we make use of mathematical programming (MP) techniques [12]. From our point of view, MP techniques provide a good tool to model this problem class as well as they allow for a common set of standard solution techniques for the different problem incarnations. In the following MP formulations of MPRASE problem incarnations we try to illustrate this point.

2.3.1 The Single Provider Problem (SPP)

Let us first look at the single provider allocation problem (SPP). The customer has capacity demands b_t that must be fully satisfied at every discrete time interval $t = 1,...,T$. As the edge model is deterministic, the demand is known in advance for all periods. Capacity is requested from a single provider who is charging a fixed costs f_t for every allocation and variable allocation costs c_t per reserved capacity unit and period. A new allocation is constituted by a change in the allocated capacity. Allocated capacity is available in the period the allocation is made and in all subsequent periods until the next allocation is made. Note that the allocated and not the really used capacity causes the costs. Using two types of variables and a number of parameters, this problem can be formulated as model M1.

M1 Single Provider Problem - SPP

Variables:

x_t	Amount of reserved capacity in period $t = 1,..., T$.
z_t	Binary variable, 1 if a allocation is made at beginning of period $t = 1,...,T$ and 0 otherwise.

Parameters:

b_t	Demanded capacity in period $t = 1,...,T$. Demand is assumed to be greater than 0.
f_t	Fixed allocation costs, costs per allocation. We assume positive costs ($f_t > 0$).
c_t	Variable allocation costs, costs per reserved capacity unit per period.
x_0	Allocation level before the beginning of the first period.
M	M is a sufficiently high number (e.g., max $\{b_t\}$).

Minimize
$$\sum_{t=1}^{T} f_t z_t + \sum_{t=1}^{T} c_t x_t \tag{1}$$
subject to
$$x_t \geq b_t \qquad \forall t = 1, ..., T \tag{2}$$
$$x_t - x_{t-1} \leq M \cdot z_t \qquad \forall t = 1, ..., T \tag{3}$$
$$x_{t-1} - x_t \leq M \cdot z_t \qquad \forall t = 1, ..., T \tag{4}$$
$$z_t \in \{0, 1\} \qquad \forall t = 1, ..., T \tag{5}$$

The objective function (1) minimizes total costs. (2) ensures that demand is fully satisfied in each period. (3) and (4) force z_t to one whenever x_t and x_{t-1} differ, i.e., a new resource allocation takes place. Note that z_t will be set to 0 in all other cases automatically because of the non-negative entry f_t in the objective function.

So far we have only assumed costs for the reserved and not for the really used capacity. We could additionally introduce costs for the really used capacity but as demand is fixed these costs would effectively be a constant in our objective function and can therefore be eliminated without influencing the solution.

2.3.2 Multi-Provider Allocation Problems

In this section, we present the multi-provider allocation problem in an uncapacitated and a capacitated version. The single provider allocation problem presented in Section 2.3.1 is a subproblem of this with the number of provider J =

1. Let us assume that there is more than one provider offering capacity to the customer. We assign index $j = 1, ..., J$ to the different providers. Based on M1 we can model this problem with M2.

M2 Multi-Provider Allocation Problem - MPP

Variables:

x_{jt} Amount of allocated capacity in interval t from provider j.

z_{jt} 1 if an allocation for provider j is made at the beginning of period t and 0 otherwise.

d_{jt} 1 if allocation for provider j drops to 0 in interval t and 0 otherwise.

Parameters:

b_t Demanded capacity in interval $t = 1, ..., T$. Demand must be fully satisfied in each period.

f_{jt} Fixed allocation costs, i.e., cost for an allocation in period t from provider j, we assume $f_{jt} > 0$.

c_{jt} Variable costs, i.e., costs per capacity unit per period (specific per provider and period).

x_{j0} Allocation level before the beginning of the first planning period.

Minimize
$$\sum_{j=1}^{J} \sum_{t=1}^{T} f_{jt}(z_{jt} - d_{jt}) + \sum_{j=1}^{J} \sum_{t=1}^{T} c_{jt} x_{jt} \qquad (6)$$

subject to

$$\sum_{j=1}^{J} x_{jt} \geq b_t \qquad \forall t = 1, ..., T \qquad (7)$$

$$x_{jt} - x_{j(t-1)} \leq M \cdot z_{jt} \qquad \forall j = 1, ..., J, \forall t = 1, ..., T \qquad (8)$$

$$x_{j(t-1)} - x_{jt} \leq M \cdot z_{jt} \qquad \forall j = 1, ..., J, \forall t = 1, ..., T \qquad (9)$$

$$d_{jt} + \varepsilon x_{jt} \leq 1 \qquad \forall j = 1, ..., J, \forall t = 1, ..., T \qquad (10)$$

$$L(x_{jt} + x_{j(t-1)}) \geq d_{jt} \qquad \forall j = 1, ..., J, \forall t = 1, ..., T \qquad (11)$$

$$d_{jt} \in \{0, 1\} \qquad \forall j = 1, ..., J, \forall t = 1, ..., T \qquad (12)$$

$$z_{jt} \in \{0, 1\} \qquad \forall j = 1, ..., J, \forall t = 1, ..., T \qquad (13)$$

$$x_{jt} \geq 0 \qquad \forall j = 1, ..., J, \forall t = 1, ..., T \qquad (14)$$

M3 Capacitated Multi-Provider Allocation Problem - cMPP

Minimize (6)

subject to (7)-(14) and

$$x_{jt} \leq k_{jt} \qquad \forall j = 1, ..., J, \forall t = 1, ..., T \qquad (15)$$

This model mainly differs from M1 in the additional index j. Furthermore, we now have to model the case that in a certain period no capacity is allocated at a certain provider. This is captured by the introduction of demand defect variables, d_{jt}, and the constraints (10) and (11). Here, ε needs to be chosen small, e.g., $\varepsilon = \frac{1}{\max\{b_t\}}$, whereas M and L need to be chosen large, e.g., $M = \max\{b_t\}$ and $L = \frac{1}{\min\{b_t | b_t > 0\}}$.

In the next step we use additional parameters, k_{jt}, to model by (15) that each provider j can offer only a limited amount of resources k_{jt} in period t. This leads to model M3, the capacitated MPP (cMPP).

2.3.3 Single Provider Problems with Flexible Demand

Above, we have assumed that the provider(s) had to satisfy customers' demand at all times. Now we describe two related SPP formulations where this is not the case, i.e., we have a different customer model. In the first problem, demand can be dissatisfied (leading to degraded quality for the customer), yet for dissatisfying demand the intermediary has to take penalty costs into account. Those can be either real costs or opportunity costs because he risks to loose customers.

A related problem is then formulated in which we explicitly assume that the demand faced by the intermediary is the superposition of several customers' demands. The intermediary has the freedom to choose among those customers whose demands it accepts and fully satisfies and whose not, i.e., it exerts admission control. Note that the model with degraded quality (which is easier) can also be used as an approximation for the problem with admission control if the number of customers is high enough (dissatisfying means not to accept all customers and the penalty costs are estimates of the lost profit).

Let us first turn to the problem with degraded quality. It is based on M1 and adds another variable, s_t, that represents how much the allocated amount of capacity remains below the demand in period t. In the objective function s_t is weighted with penalty costs c_t^s. The problem only makes sense if the penalty costs c_t^s are higher than the variable costs c_t because otherwise it would be optimal to never allocate anything. The MP formulation is given by M4.

M4 Single Provider Problem with Degraded Quality - SPP$_{DQ}$

Variables:

s_t Unsatisfied demand in period t.

Parameters:

c_t^s Penalty costs per unit of dissatisfied demand in period t.

Minimize $$\sum_{t=1}^{T} f_t z_t + \sum_{t=1}^{T} c_t x_t + \sum_{t=1}^{T} c^s{}_t s_t \qquad (16)$$

subject to (4), (5) and

$$x_t + s_t \geq b_t \qquad \forall t = 1, ..., T \qquad (17)$$

M4 differs from M1 in two ways: objective function (16) now includes the penalty costs and constraint (17) allows to dissatisfy demand by setting s_t to a positive value.

Let us now have a look at the problem with admission control. We introduce index i for identifying the customers. Each accepted customer obtains its demanded capacity and pays a price p_i. The resulting model is given by M5. Objective function (18) maximizes profit, constraint (19) ensures that the demand of all accepted customers is fully satisfied in each period. In M5, it is sensible to assume the intermediary is located at the provider whereas for the other

M5 Single Provider Problem with Admission Control - SPP$_{AC}$

Variables:

a_i 1 if customer *i* is accepted, i.e., if his demand is fully satisfied or 0 otherwise.

Parameters:

b_{it} Demanded capacity from customer *i* = 1, ..., *I* in interval *t* = 1, ..., *T*.

$$\text{Maximize} \quad \sum_{i=1}^{I} p_i a_i - \sum_{t=1}^{T} f_t z_t - \sum_{t=1}^{T} c_t x_t \tag{18}$$

subject to (3)-(5) and

$$x_t \geq \sum_{i=1}^{I} a_i \cdot b_{it} \qquad \forall t = 1, ..., T \tag{19}$$

$$a_i \in \{0, 1\} \qquad \forall i = 1, ..., I \tag{20}$$

problems described above the intermediary’s location would have naturally been at the customer. Note, however, that there might also be some third party taking the role of the intermediary, e.g., as a service to the customer.

2.3.4 Single Provider Token Bucket Allocation Problem

So far traffic was described only by one parameter, the capacity (for example bandwidth). Multimedia traffic normally includes a kind of burstiness and is therefore described better described by other models. A commonly used model for describing multimedia streams is the token bucket filter. We try to reformulate Model M1 in order to use token bucket as resource model.

Token Bucket characteristics One common way to describe bandwidth and burst characteristics of 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 bytes in one period the sender must own n tokens. The sender starts with no 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 ashis hardware is able to), but over a sufficiently long interval, he can't send more than r bytes per second.

A common way to picture the token bucket characteristics in continuous time is depicted in figure 3. 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 must remain below the depicted curve that starts at B and has a slope of r at all times t_0.

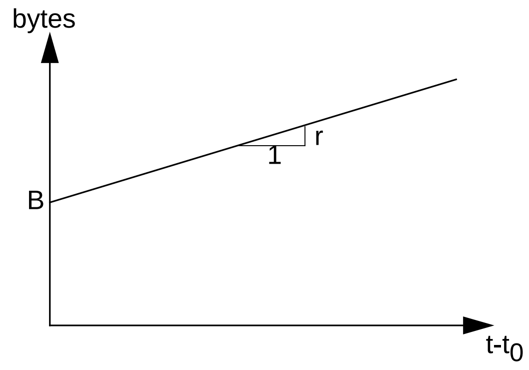


Figure 3: Token-Bucket curve

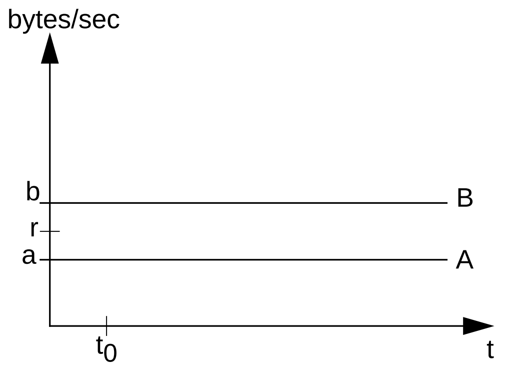


Figure 4: Two example data flows

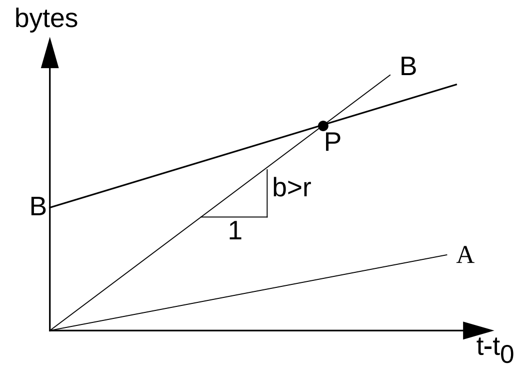


Figure 5: Flow B running out of tokens at point P

Fig. 4 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 3. A transformation[1] of fig. 4 into fig. 3 is shown in fig. 5 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.

For modelling puposes we use discrete time intervals. Fig. 3 corresponds to Fig. 6, rate r refers to one period now (instead of one second). In the first period you are allowed to send B bytes plus r bytes, because you get r tokens that period. Fig. 7 shows two example flows that are transformed for the critical interval t_0 (see fig. 8). As we can see, flow C equals the token-bucket curve while flow D remains below it for all the time.

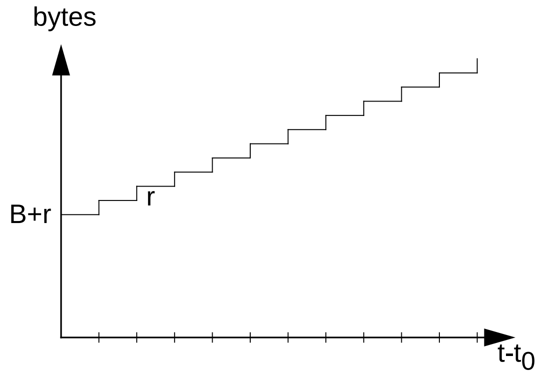


Figure 6: Token-Bucket curve in discrete time

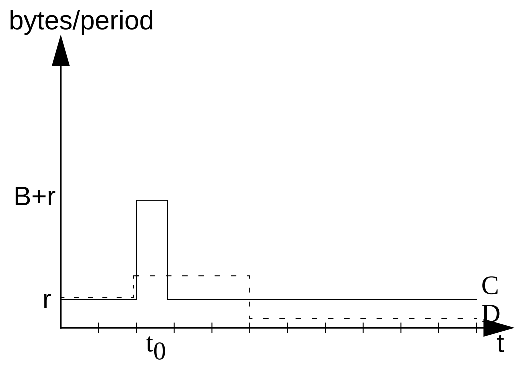


Figure 7: Two example data flows in discrete time

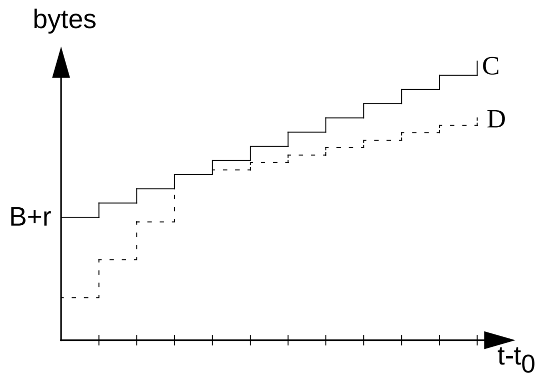


Figure 8: Aggregation of the two data flows

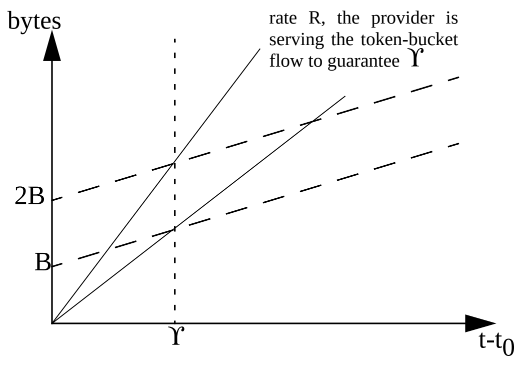
Pricing Before we start to model the reservation problem using token buckets characteristics let’s first think if linear prices for the rate and the bucket depth are reasonable. It is reasonable to assume linear prices for bandwidth, because otherwise it would be possible to arbitrage the provider . This is why we have assumed linear prices for the reserved capacity in the models above.

But is it also reasonable to assume linear prices for the bucket-depth? To answer this question we look at the following type of contract between us and our provider:

„I can reserve variable bandwidth by making reservations for my data flow. The flow must be described with the two parameters of the token-bucket model. For each reservation I have to pay a fixed reservation price and I have to pay a certain price each period depending on the parameters of the token bucket. The provider is offering a guaranteed service, that means he is guaranteeing that none of my packets are dropped (as long as they comply with the token-bucket model) and every packet reaches it’s destination 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 9. 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.

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


bytes
rate R, the provider is serving the token-bucket flow to guarantee Υ
2B
B
Υ
t-t0
Figure 9: Bucket depth and needed bandwidth

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.

Model M6 is based on M1 but instead of a one dimensional capacity a two-dimensional buffered resource model based on a token-bucket is used. Equations (21)/(22) minimize the total costs. Note that the reserved and not the used bucket rate and height has to be paid.The variables y_t are included in the objective function as an incentive to refill the bucket via (27) as quickly as possible. (23) ensures that the demand is met each period, if the rate is not enough, y_t extra tokens have to be used. (24) and (25) in combination with (3) and (4) force z_t to one whenever h_t and/or x_t are changed, that is when a new allocation is made. (27) ensures that used tokens are accounted for correctly.

M6 (Single Provider Token Bucket Allocation Problem)

Variables:

y_t Excess capacity in period $t = 1, ..., T$.

l_t Total amount of excess tokens in period $t = 1, ..., T$.

h_t Bucket depth in period $t = 1, ..., T$.

Parameters:

c_t^r Rate costs, costs per reserved capacity-unit per period (assuming linear cost strukcture).

c_t^b Bucket depth costs, costs per bucket depth unit per period (assuming linear cost structure).

l_0 Amount of used tokens at the beginning of the first period.

ε Small number > 0, e.g., .

Minimize $$\sum_{t=1}^{T} c(t, h_t, x_t, z_t) + \sum_{t=1}^{T} \varepsilon y_t \qquad (21)$$

or, assuming a linear cost structure

Minimize $$\sum_{t=1}^{T} f_t z_t + \sum_{t=1}^{T} c_t^r x_t + \sum_{t=1}^{T} c_t^h h_t + \sum_{t=1}^{T} \varepsilon y_t \qquad (22)$$

subject to (3)-(5) and

$$x_t + y_t \geq b_t \qquad \forall t = 1, ..., T \qquad (23)$$

$$h_t - h_{t-1} \leq M \cdot z_t \qquad \forall t = 1, ..., T \qquad (24)$$

$$h_{t-1} - h_t \leq M \cdot z_t \qquad \forall t = 1, ..., T \qquad (25)$$

$$l_t \leq h_t \qquad \forall t = 1, ..., T \qquad (26)$$

$$l_t = l_{t-1} + y_t \qquad \forall t = 1, ..., T \qquad (27)$$

$$y_t \in R \qquad \forall t = 1, ..., T \qquad (28)$$

$$h_t \geq 0, \ l_t \geq 0 \qquad \forall t = 1, ..., T \qquad (29)$$

3 Tackling the Single Provider Problem

After the presentation of the general MPRASE framework, we now want to investigate different solution techniques for the simplest possible problem incarnation: the uncapacitated single provider problem at a deterministic edge. From simulative experiments as well as theoretical observations it turns out that even this problem is not absolutely trivial although we are able to devise reasonably efficient exact solutions as well as computationally inexpensive, yet qualitatively satisfying heuristic techniques. Besides the value of having a solution for the SPP, these techniques are also the base components for devising techniques for more advanced problems. This will be demonstrated in Section 4 where we tackle the MPP by reusing solution techniques for the SPP.

3.1 Optimality Conditions

The optimal solution of M1 must adhere to the following condition (30):

$$\forall t \in [1, T] : x_t = max[b_\tau | \tau \in [\underline{\upsilon}(t), \bar{\upsilon}(t)]] \tag{30}$$

$$\underline{\upsilon}(t) = max[\tau | (z_\tau = 1) \wedge (\tau \le t)] \tag{31}$$

$$\bar{\upsilon}(t) = min[\tau | (z_\tau = 1) \wedge (\tau > t)] \tag{32}$$

The interval $[\underline{\upsilon}(t), \bar{\upsilon}(t)]$ is the interval for which the allocation that includes period t is made.(30) effectively means, that the allocated amount of capacity is exactly equal to the maximum amount of demand in the periods for which the allocation is made. It clearly does not make sense to allocate more capacity than the maximum requested and it is forbidden to reserve less.

If we assume that all f = f_t are equal, which is a reasonable assumption considering the short timescale we are planning for, the following condition (33) must also be met in an optimal solution:

$$\forall t \in [1, T] : x_t = b_t \Rightarrow z_t = 0 \tag{33}$$

This means, that if we can satisfy the demand in period t exactly, it does not make sense to make a new reservation to buy more capacity now for later periods, because this can always be done at later periods for the same fixed costs f.

3.2 Exact Algorithms

At first we want to look at techniques that guarantee to produce an optimal solution for the SPP.

3.2.1 Full Enumeration

In every period a allocation can be made or not. Once the decision is made in which periods allocations are made, it is very easy to calculate the cost minimal allocation levels x_t using the optimalitz conditions from Section 3.1. But as there are 2^T possible solutions the algorithm needs exponential time.

3.2.2 Branch and Bound with Linear Programming (LP) Relaxation

A standard approach to solve the single provider problem SPP is to use a mixed integer problem solver in order to solve model M1. A typical algorithm for solving a mixed integer LP model is a branch and bound algorithm that uses the LP relaxed problem M1’ of M1:

M1’ LP Relaxation of M1 (SPP)

The binary condition (5) is dropped from M1 and replaced by

$$0 \le z_t \le 1 \qquad \forall t = 1, ..., T \qquad (34)$$

Bounding The resulting problem can be easily solved with the simplex algorithm. The solution of M1’ is a lower bound to the optimal solution of M1. To get a very good (that is very high) lower bound, parameter M must be as low as possible. If M is very high, z_t can be very close to zero while still making new allocations every period resulting in a very bad lower bound.

A good value for M is the maximum demand in all periods as this is the maximal difference between any two x_t:

$$M = max\{b_t | t \in [1, T]\}$$

Note that we can also calculate an upper bound each iteration.

Branching Branching can be done by fixing the highest not yet fixed z_t to 1 in the first and to 0 in the second subproblem. Branching can also be done by fixing the z_t with the highest (lowest) difference between b_{t-1} and b_t. This is done because high (low) differences make a new allocation more (less) probable resulting in the subproblem prohibiting (forcing) the new allocation beeing ruled out more often.

An example problem with only 50 periods took already 33 minutes to be solved[2]. Problems with more than 100 periods could not be solved within several days. The reason for this is that the structure of the problem does not make it very amenable to branch and bound algorithms since z_t are often set to very low values greater 0 resulting in a vast underestimation of fixed costs which leads to very loose bounds. Therefore, we strived for more efficient, yet still exact algorithms for the SPP.

3.2.3 Shortest Path (SP) Algorithm

The SPP can be transformed into a shortest path problem which can then be solved with well-known algorithms as, e.g., Dijkstra’s Algorithm [13]. The graph for the shortest path problem consists of T+1 vertices $\{t_1,..., t_{T+1}\}$, one for each period (numbered in ascending order) and one final state vertex. Each edge (t_i, t_j) represents an allocation from period t_i to t_j. For each vertex t_i there must be an edge to all vertices t_j with $j \ge i$. This results in the graph representing

[2] All experiments have been performed on a 400 MHz Pentium II processor.

all possible multi-period allocations. Each edge is assigned the costs of a single allocation, which is calculated as follows:

Let (t_1, t_2) be the edge from vertex t_1 to vertex t_2. The costs (or length) of edge (t_1, t_2) are defined as

$$C(t_1, t_2) = f_{t_1} + \sum_{\tau = t_1}^{t_2} c_\tau \cdot max(b_t | t \in \{t_1, ..., t_2\}) . \tag{35}$$

The number of edges is $\frac{1}{2}T(T+1)$, resulting in a complexity for creating the graph of $O(T^3)$. Dijkstra's shortest path algorithm then requires $O(T^2)$ steps to find the optimal solution, i.e., we obtain an overall complexity of $O(T^3)$.

3.2.4 Dynamic Programming (DP)

An $O(T^2)$ approach which also guarantees to find an optimal solution is the dynamic programming algorithm given in Figure 10. The function $C(t_1, t_2)$ is the same as in (35).

```
Preparation:
    Prepare an empty array cMin and an empty array pred, each with T entries.
Start:
    $cMin(1) = C(t_1, t_1)$
    $pred(1) = 1$
Iteration $t = 2, ..., T$:
    $cMin(t) = \min\{C(i, t) + cMin(i-1) \mid i = 1, ..., t\}$
    $pred(t) = \mathrm{argmin}\{C(i, t) + cMin(i-1) \mid i = 1, ..., t\}$
Result:
    $cMin(T+1)$ contains the minimal costs while array pred stores the hops towards that solution.
```

Figure 10: Dynamic programming algorithm for SPP.

This algorithm exploits the structure of the problem which causes $C(t_y, t_x) \le C(t_y, t_{x+1}) \;\; \forall (t_y, t_x) | x > y$. Hence, the complexity of DP can be reduced to $O(T^2)$ in comparison to SP. A further advantage of DP over SP is that it only needs an $O(T)$ instead of $O(T^2)$ sized table to store intermediate solutions.

3.2.5 Assessment of Execution Times

Table 2 shows the execution times for all of the exact algorithms for two differently sized problem instances.

Algorithm	B&B	SP	DP
T=50	*1920.7*	*0.0046*	*0.0026*
T=1000	*not available*	*38.2*	*9.0*

Table 2: Execution times for exact SPP algorithms (in sec).

3.3 Heuristics

While the last section introduced exact solutions for the SPP, which while they provided fairly good performance still required a certain computational effort that might be prohibitive in scenarios where there is either a large number of periods to be planned for or where there is only an extremely limited amount of time available for computation as, e.g., if the resource allocation is done in response to signalling messages and thus affects setup latencies. Therefore, we now want to investigate heuristic techniques which do not guarantee an optimal solution but allow very fast allocation decisions. A further reason for investigating heuristics becomes obvious when we extend the SPP techniques towards the MPP in Section 4 where we then need to solve a potentially large number of SPPs.

3.3.1 LP Heuristic (LH)

The LP heuristic is solving the LP relaxation M1’ of Section 3.2.2 to determine the amount of allocated capacity. After solving M1’ (using the simplex algorithm), any $z_t \neq 0$ is set to 1 wherever necessary (that is, where x_t and x_{t-1} differ). This leads to a relative high number of allocations since fixed costs are systematically underestimated by allowing continuous z_t. To improve the results of LH the parameters f_t can be replaced with $f_t' = a_t f_t$ with $a_t > 1$.

3.3.2 Merge Heuristic (MH)

The merge heuristic starts with a separate allocation for each period and then tries to merge two successive allocations into one if the saved fixed costs of the allocation are less than the waste of variable costs (see Figure 11 for an illustration of this). A formal description for the merge heuristic is given in Figure 12.

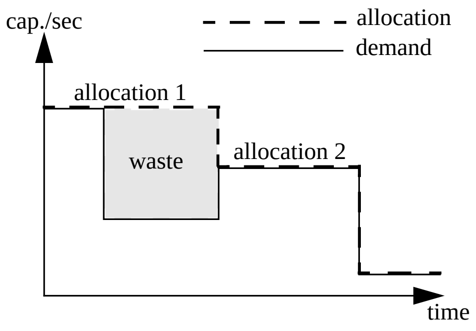


Figure 11: Waste of variable costs.

3.3.3 Split Heuristic (SH)

The split heuristic starts with a single allocation and then tries for all periods to split existing allocations if the fixed costs for the new allocation are less than the saved waste of variable costs. A formal description of SH is given in Figure 13.

```
Assumptions:
    $x_0 = 0$ and $b_0 \neq 0$ Start:
    $z_t = 1$ and $x_t = b_t$ for all $t = 1, ..., T$
Iteration $n$:
  For each $t_0$ in $[1, T-1]$ with $z_{t_0} = 1$ try:
    $t_1$ is the next period after $t_0$ with $z_{t_1} = 1$
    $t_2$ is the next period after $t_1$ with $z_{t_2} = 1$
    $B_{t_0} = max\{b_t | t \in [t_0, t_1 - 1]\}$
    $B_{t_1} = max\{b_t | t \in [t_1, t_2 - 1]\}$
    if ($B_{t_1} < B_{t_0}$) then
        $\Delta = B_{t_0} - B_{t_1}$ and $\Lambda = [t_1, t_2-1]$
    else $\Delta = B_{t_1} - B_{t_0}$ and $\Lambda = [t_0, t_1-1]$
    if $f_{t_1} \geq \sum_{\tau \in \Lambda} c_\tau \Delta$ then
        merge $[t_0, t_1-1]$ and $[t_1, t_2-1]$

Stop:
    If no merges during last iteration
Merging:
    set $z_{t_1} = 0$ and $x_t = max(b_{t_0}, b_{t_1})$ $\forall t \in \{t_0, ..., t_2\}$
```

Figure 12: Merge heuristic.

```
Start:
  $x_t = max(b_t | t = 1, ..., T)$ for $t = 1, ..., T$
  $z_1 = 1$ and $z_t = 0$ for $t = 2, ..., T$
Iteration $n$:
  For each $t_1$ in $[2, T]$ with $z_{t_1} = 0$ try:
    $t_0$ is the latest period before $t_1$ with $z_{t_1} = 1$
    $t_2$ is the next period after $t_1$ with $z_{t_2} = 1$
    $B_{t_0} = max\{b_t | t \in [t_0, t_1 - 1]\}$
    $B_{t_1} = max\{b_t | t \in [t_1, t_2 - 1]\}$
    if ($B_{t_1} < B_{t_0}$) then $\Delta = B_{t_0} - B_{t_1}$ and $\Lambda = [t_1, t_2-1]$
    else $\Delta = B_{t_1} - B_{t_0}$ and $\Lambda = [t_0, t_1-1]$
    if $f_{t_1} \leq \sum_{\tau \in \Lambda} c_\tau \Delta$ then
        split $[t_0, t_2-1]$ into $[t_0, t_1-1]$ and $[t_1, t_2-1]$

Stop:
  If no splits during last iteration
Splitting:
  set $z_{t_1} = 1$ and $x_t = max\{b_t | t \in \Lambda\}$ $\forall t \in \Lambda$.
```

Figure 13: Split heuristic.

3.3.4 Incomplete Branch and Bound

3.3.5 Another heuristic can be easily implemented by stopping the branch and bound algorithm if it takes to long and to use the best solution found so far.**Combined Heuristics (CH[x,y])**

3.3.6 The merge and split heuristics can also be used to further improve the results of other heuristics. In our simulations we therefore iterated through merge and split in sequence until no further improvement could be achieved (CH[MH, SH]). Moreover, we also tried the combination of merge and split based on the result of the LP heuristic

(CH[LP,MH,SH]).**Simulations for Qualitative Assessment of the Heuristics**

In order to evaluate the performance of the heuristics we ran a simulation over 100 random problem instances, each with T=1000, fixed costs $f_t \in$ [200,800] drawn from a uniform random distribution once and then set equal for all T periods. Variable costs c_t are drawn from [3,5] and remain equal for p periods; p is drawn from [10,20].

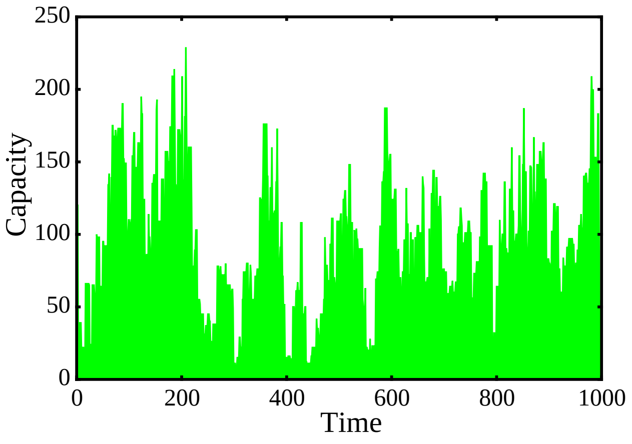


Figure 14: Sample capacity demand curve.

The demand is calculated by superposing a number of requests (for example representing individual requests from several users) with their interarrival time modelled by a poisson distribution (λ = 4) and their duration modelled by an exponential distribution (μ = 20) [3]. For calculating the requests' capacity demand we draw from a uniform random distribution from one out of three possible intervals [2,8], [10,20] and [35,50] representing small, medium and high capacity requests. The interval itself is selected for each request with a probability of 40%, 30% and 30%. Figure 14 shows a sample problem generated in this way.

Table 3 shows the results generated by the simulations. Here, allocation length denotes the average duration of a single allocation and waste is the total waste of variable costs for a single SPP instance (as illustrated in Figure 11).

Algorithm	Costs	Relative deviation from optimum costs				Allocation length	Waste	Time (sec)
	av	*av*	*stddev*	*min*	*max*	*av*	*av*	*av*
Optimum (DP)	*452304*	*n.a.*	*n.a.*	*n.a.*	*n.a.*	*9.43*	*36515*	*9.000*
PH	*1010199*	*123.81%*	*32.96%*	*58.97%*	*221.73%*	*1000.00*	*645804*	*< 0.001*
MH	*474027*	*4.79%*	*1.07%*	*2.05%*	*7.15%*	*10.65*	*64257*	*0.002*
SH	*568759*	*25.93%*	*10.43%*	*12.96%*	*73.65%*	*3.72*	*63295*	*0.010*
LP	*554317*	*22.34%*	*8.37%*	*6.12%*	*39.07%*	*2.62*	*424*	*0.452*
CH[MH, SH]	*469723*	*3.85%*	*0.74%*	*1.80%*	*5.34%*	*9.81*	*56064*	*0.005*
CH[LP, MH, SH]	*460404*	*1.77%*	*0.70%*	*0.39%*	*3.75%*	*8.93*	*41918*	*0.452*

Table 3: SPP simulation results.

As a very simple alternative heuristic and to have a reference value we also used what we called the peak heuristic (PH) which makes a single allocation with the highest capacity demand over all periods. Expectedly, PH performed very poorly compared to the other techniques. A much better performance at very low execution time is achieved by the merge heuristic (MH): on average it imposes less than 5% additional costs relative to the optimum and reduces ex-

[3] We have to admit that parameter choice is rather arbitrary (albeit sensible) due to lack of empirical data. However, we have experimented with other values without changing the results in a significant manner.

ecution time by a factor of 4500. The conceptually very similar split heuristic (SH) is considerably less effective. Looking at the allocation length shows the reason: it overdoes its job by splitting too often, resulting in too short allocation lengths and thus incurring fixed costs more often although waste of variable costs is roughly equal to MH.

The LP heuristic performs only marginally better than SH, although it consumes considerably more time. This is due to its characteristic of underestimating fixed costs which is also expressed in a very low waste and small allocation lengths.

Next, let us see how these results may be improved by the combination of heuristics as described in Section 3.3.5. The combination of MH and SH leads expectedly to better results than the techniques in isolation. Yet, even better results can be achieved by integrating LP with MH and SH.

In conclusion, the best results are achieved by CH[LP,MH,SH], yet the most attractive trade-off between cost performance and execution time is probably achieved by MH or CH[MH,SH].

4 Extension to the Multi-Provider Problem

After the rather extensive treatment of the SPP, we now want to extend our investigations along the provider model by assuming more than one provider, i.e., we move to the MPP. Note that the MPP can also model a single provider that offers multiple tariffs among which can be chosen, e.g., tariffs comprising different fixed and variable costs which do not dominate each other. In line with our argument for taking an integrated view on MPRASE problems we approach the MPP by trying to take advantage from the techniques developed for the SPP.

In Section 2.3.2, we have modelled two different versions of the MPP, an uncapacitated and a capacitated one. Here, we want to take a look at both of them. The uncapacitated MPP represents a situation where a customer’s demand is relatively small compared to the provider’s supply such that the resulting problem consists mainly in the selection of the cheapest provider. The capacitated MPP (cMPP), on the other hand, rather deals with a good mixing of providers to achieve low total costs.

Note that the problem complexity of MPP is much higher than that of SPP. First, the demand of each period can be satisfied by $2^{J}-1$ different combinations of providers and second, if two or more providers are selected to satisfy the demand of one period there is a high number of sensible shares between these. This higher complexity is also illustrated by the execution times of applying the standard branch and bound solver to model M2. A small MPP with T=20 and J=4 already took 1920.8 seconds to solve while the corresponding SPP with T=20 only took 1.2 seconds. For any larger MPPs execution times were no longer reasonable. With this complexity in mind we go directly for heuristics and try to use our knowledge about the SPP.

4.1 SPP-Based Heuristics for the Uncapacitated MPP

4.1.1 Static Cheapest Provider Heuristic (SCPH)

A straightforward approach to tackle the uncapacitated MPP is to transform it into J SPPs, one for each provider and each with the full demand. The SPPs can then be solved by any of the algorithms discussed in Section 3. After solving the J SPPs we select the provider of the SPP with the least costs. That means we obtain a solution where one provider is used for all periods.

In our simulations, we use DP to solve the SPPs because it yields the optimal results for the SPP at affordable execution time. Alternatively, we use the peak heuristic to see how non-optimal solutions for the SPP will affect the result for the MPP.

4.1.2 Dynamic Cheapest Provider Heuristic (DCPH)

One drawback of SCPH is that it does not allow provider changes. Using a technique similar to the dynamic programming algorithm from Section 3.2.4 we can eliminate this characteristic of the SCPH. The resulting algorithm is called dynamic cheapest provider heuristic (DCPH). This is also illustrated in Figure 15.

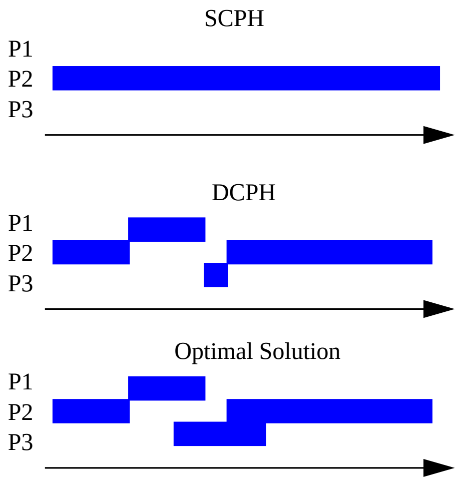


Figure 15: Provider usage of the different algorithms for the uncapacitated MPP.

We use the same algorithm as in Section 3.2.4, but the minimal costs $C(t_1, t_2)$ for satisfying the demand between two periods t_1 and t_2 are obtained by solving J independent SPPs for the interval $[t_1, t_2]$ and choosing the cheapest provider. Unlike the DP algorithm from Section 3.2.4, this algorithm does not necessarily lead to the optimal result as it does not allow for a constellation as depicted for the optimal solution in Figure 15. Again, we have the freedom of selecting any of the SPP algorithms for solving the sub-SPPs. In our simulations, we choose again DP and PH.

4.2 Adaptation of the Heuristics for the Capacitated MPP

If the capacity of one provider is not enough to satisfy the whole demand we can no longer simply select a single provider in SCPH and DCPH but have to combine several providers. We do this by first cropping the demands in each SPP to the capacity of the according provider. We then solve the SPPs for all J providers and select the provider that has the minimum costs per satisfied demand. The overall demand is then reduced by the capacity served by the selected provider and the procedure is repeated until no more demand remains unsatisfied. Example allocations are shown in Figure 16.

Please note that the non-zero demand assumption in section 2.3.1 can now no longer be held and model M1 as well as the heuristics of Section 3.3 had to be adapted to cope with periods of no demand.

4.3 Other Heuristics for the MPP

4.3.1 LP Heuristic (LP)

Of course, we can again use the results of the LP relaxation for M2 and M3 to obtain a solution for MPP/cMPP.

4.3.2 Merge Heuristic (MH)

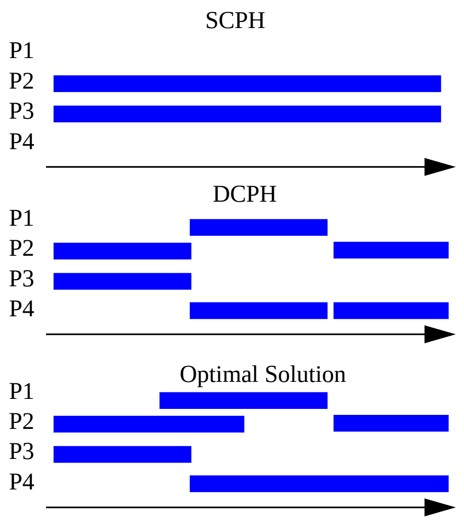


Figure 16: Provider usage of the different algorithms for the capacitated MPP.

We adapted the merge heuristic to the multi-provider case and to the capacity constraints and combined it with DCPH and LP in order to investigate whether it can improve their solutions.

4.4 Simulations for Qualitative Assessment of the Heuristics

In order to evaluate the MPP heuristics described above we ran a simulation over 50 MPP instances similar to the simulations in Section 3. Because of the much higher computational complexity we reduced the number of periods T from 1000 to 100. We used 10 providers and three different levels of capacity: first the uncapacitated MPP, second a cMPP with the capacity of each provider drawn from [30%, 50%] of the maximum demand over all periods, and third a cMPP with provider capacity drawn from [15%, 35%].

Algorithm	Costs	Allocation Length	Waste	Total Number of Providers	Av. Number of Providers
SCPH (PH)	*59563*	*100.00*	*350954*	*1.00*	*1.00*
SCPH (DP)	*36428*	*8.23*	*31271*	*1.00*	*1.00*
DCPH (PH)	*31999*	*7.62*	*29751*	*5.10*	*1.00*
DCPH (DP)	*31999*	*7.62*	*29751*	*5.10*	*1.00*
CH[DCPH (PH), MH]	*31999*	*7.62*	*29751*	*5.10*	*1.00*
LP	*39165*	*2.52*	*35*	*3.28*	*1.27*
CH[LP, MH]	*35246*	*5.55*	*27269*	*3.28*	*1.27*

Table 4: Results for the uncapacitated MPP.

The results for the uncapacitated MPP simulations are displayed in Table 4. Here, the total number of providers denotes how many of the providers were selected at least once by a heuristic whereas average number of providers expresses how many providers were on average simultaneously active.

DCPH is obviously and expectedly significantly better than SCPH, however all DCPH-based heuristics perform identically, although the SPP heuristics are very different. Even if using the simple PH as SPP solver the results are very good (in contrast to SCPH(PH)). The explanation for this behaviour is that the use of the dynamic programming paradigm as provider selection strategy makes the use of dynamic programming as SPP strategy unnecessary.[4] Note that MH is ineffective for DCPH because within its range (i.e., only one provider at a time) the solution is already optimal. The LP heuristic shows the same behaviour as in the SPP case and can again be further improved by MH.

Algorithm	Costs	Allocation Length	Waste	Total Number of Providers	Av. Number of Providers
SCPH (PH)	*65548*	*32.17*	*320415*	*3.14*	*3.14*
SCPH (DP)	*38890*	*7.64*	*31421*	*3.04*	*1.90*
DCPH (PH)	*38048*	*5.63*	*45981*	*6.68*	*1.88*
DCPH (DP)	*35650*	*6.72*	*29182*	*5.52*	*1.79*
CH[DCPH (PH), MH]	*36990*	*7.21*	*49292*	*6.64*	*1.88*
LP	*44334*	*2.21*	*10*	*4.96*	*2.05*
CH[LP, MH]	*38910*	*5.04*	*27344*	*4.96*	*2.05*

Table 5: Results for the capacitated MPP (high capacity providers).

Algorithm	Costs	Allocation Length	Waste	Total Number of Providers	Av. Number of Providers
SCPH (PH)	*62219*	*19.60*	*348896*	*5.18*	*5.18*
SCPH (DP)	*37084*	*7.42*	*34826*	*4.68*	*2.77*
DCPH (PH)	*39105*	*5.22*	*58985*	*7.08*	*2.75*
DCPH (DP)	*34976*	*6.52*	*29759*	*6.26*	*2.58*
CH[DCPH (PH), MH]	*37165*	*7.61*	*63301*	*7.00*	*2.75*
LP	*45167*	*2.04*	*78*	*5.72*	*2.75*
CH[LP, MH]	*37860*	*5.25*	*32848*	*5.72*	*2.75*

Table 6: Results for the capacitated MPP (low capacity providers).

In Table 5 and 6 the results for the cMPP with high respectively low capacity providers are given. Now DCPH with DP is significantly better than with PH. Since more providers are involved at the same time there is now potential for more intelligent SPP strategies such as DP. Not surprisingly, the number of providers in the low capacity provider case is higher.

Algorithm	Execution Time (sec)
SCPH (PH)	*0.010*
SCPH (DP)	*0.397*
DCPH (PH)	*0.689*
DCPH (DP)	*243.073*
CH[DCPH (PH), MH]	*0.689*
LP	*2.323*
CH[LP, MH]	*2.323*

Table 7: Execution times of MPP heuristics.

To make recommendations for the MPP heuristics we also have to take into account the execution times of the different alternatives as given in Table 7. While DCPH(DP) exhibits the best cost performance it needs considerably more time than the other heuristics. If execution time plays a crucial role then SCPH(DP) is a good choice.

[4] This is due to the fact that under the further constraint of using only one provider at a time DCPH is an exact algorithm.

5 Related Work

It is difficult to find directly related work since our MPRASE approach is very general and other work mainly treated individual MPRASE problem incarnations in isolation. Yet, interestingly, work done in the field of renegotiable services, e.g. [14,15] , arrives at very similar algorithms to calculate renegotiation schedules for stored video transmissions which furtherly emphasizes the general structure of the MPRASE framework for problems found in the domain of providing network QoS. However, the emphasis of [14,15] is on the definition of renegotiable services and not so much on algorithms and their evaluation whereas this is the main focus of our paper.

Relating to the analysis of dynamic provisioning in a multi-provider environment (in particular a DiffServ environment), the work described in [16] gives very interesting insights into the global behaviour of such a system by game-theoretic observations. Yet, our perspective is more local from a single system's point of view and how to optimize resource allocations for its purposes. It would certainly be interesting to investigate how providers using optimized resource allocation strategies as proposed in our work would interoperate from a global point of view.

In [8], we dealt with the SPP under uncertainty and made use of the deterministic edge SPP in an adaptive heuristic scheme which is based on past optimal resource allocations. This constitutes another example of utilization of the MPRASE framework in that it links different problem incarnations together. In general, MPRASE problems with an uncertain edge model show similarities to capacity management problems from other domains like air-line reservations or hotel booking in which usually yield management techniques are applicable, e.g. [17]. In contrast to yield management, however, we view the MPRASE problems from the perspective of an intermediary acting on behalf of the customer (in most models) and not so much from the provider's perspective as is the case with yield management.

6 Conclusion & Outlook

This paper has dealt with a so far largely neglected class of network QoS problems related to resource allocation at system edges over multiple time periods. We developed the MPRASE model to classify and describe this class of problems and to analyse their mutual dependencies and relationships. Next, we have established a solution framework for MPRASE based on mathematical programming models. The most basic MPRASE problem (SPP) has been dealt with extensively by developing and evaluating a variety of exact as well as heuristic techniques. The algorithms perform fast and well. We have then shown how to extend the SPP along one dimension of MPRASE (the provider model) towards the MPP and how to apply the SPP techniques to this extension. Since the MPP is a very complex problem we concentrated on the development and evaluation of heuristics. With these it has been possible to solve the MPP in an efficient way.

Many interesting issues for future work arise from our MPRASE framework. For example, it will be very interesting to investigate solution techniques for other problem incarnations with resource models that incorporate more than one dimension of capacity and to extend our models towards a stochastic edge. Also, a parameter sensitivity analysis for the problems discussed in this paper is planned as future work.

References

[1] W. Fang and L. L. Peterson. Inter-AS Traffic Patterns and Their Implication. In *Proceedings of Global Internet Workshop at GLOBECOMM'99*. IEEE, December 1999.

[2] Cooperative Association for Internet Data Analysis (CAIDA). Visualizing Internet Topology at a Macroscopic Scale, 2001. http://www.caida.org/analysis/topology/as_core_network/.

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

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

[5] U. Black. ATM: Foundation for Broadband Networks, 1995. Prentice Hall, Englewood Cliffs.

[6] J. Schmitt, M. Karsten, and R. Steinmetz. On the Aggregation of Deterministic Service Flows. *Computer Communications*, 24(1):2–18, January 2000.

[7] J. A. Cobb. Preserving Quality of Service Guarantees in spite of Flow Aggregation. In *Proceedings of the 6th International Conference on Network Protocols (ICNP'98)*, pages 90–97. IEEE, November 1998.

[8] J. Schmitt, O. Heckmann, M. Karsten, and R. Steinmetz. Decoupling Different Time Scales of Network QoS Systems, 2001. Currently under submission. ftp://ftp.kom.tu-darmstadt.de/pub/papers/SHKS00-1-paper.ps.gz.

[9] L. C. Wolf and R. Steinmetz. Concepts for Resource Reservation in Advance. *Multimedia Tools and Applications*, 4(3):255–278, May 1997. Special Issue on State of the Art in Multimedia Computing.

[10] M. Karsten, N. Berier, L. C. Wolf, and R. Steinmetz. A Policy-Based Service Specification for Resource Reservation in Advance. In *Proceedings of the International Conference on Computer Communications (ICCC'99), Tokyo, Japan*, pages 82–88, September 1999. ISBN 1-891365-05-3.

[11] O. Schelen, A. Nilsson, J. Norrgard, and S. Pink. Performance of QoS Agents for Provisioning Network Resources. In *Proceedings of the Seventh IEEE/IFIP International Workshop on Quality of Service (IWQoS'99), London, UK*, pages 17–27. IEEE/IFIP, Jun 1999. ISBN 0-7803-5671-3.

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

[13] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1(1):269–271, January 1959.

[14] M. Grossglauser and S. Keshav. RCBR: A Simple and Efficient Service for Multiple Time-Scale Traffic. *IEEE/ ACM Transactions on Networking*, 5(6):741–755, December 1997.

[15] 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.

[16] N. Semret, R. R. Liao, A. T. Campbell, and A. A. Lazar. Peering and Provisioning of Differentiated Internet Services. In *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'00)*, pages 414–420. IEEE, March 2000.

[17] R. Wollmer. An Airline Seat Management Model for a Single Leg Route when Lower Fare Classes Book First. *Operations Research*, 40(4):26–37, 1992.