Analysis of adaptive resource distribution algorithms in the framework of a dynamic DiffServ IP network

Thomas Engel, Siemens Corporate Technology, thomas.engel@mchp.siemens.de
Eugenia Nikolouzou, National University of Athens, enik@telecom.ntua.gr
Fabio Ricciato, CoRiTeL, ricciato@coritel.it
Petros Sampatakos, National University of Athens, psampa@telecom.ntua.gr

ComCon 8
June 25-29, 2001, Crete, Greece

Abstract
AQUILA, a joint European research project funded by EU, defines and implements a new Resource Control Layer (RCL) to enable dynamic end to end QoS provisioning in IP networks for QoS sensitive applications in a scalable and efficient way. The RCL controls the resource usage in an IP backbone that supports different transport services through the *Differentiated Services* (DiffServ) approach.
In the DiffServ model the Admission Control (AC) is performed by the elements at the edge of the network. To gain scalability, the AC decision must be based on locally stored information at these elements – mainly resources availability – and keep low the amount of interactions with the rest of the network. To gain efficiency, the availability of resources within the network must be dynamically tracked to some extent. To meet these twofold objective, in the framework of the AQUILA architecture a distributed provisioning architecture is proposed based on the concept of dynamical Resource Pool (RP). The RCL do not fix resource partitioning completely but retains flexibility through the concept of RPs. Hierarchical sets of RPs are used for dynamic resource re-distribution.
This paper describes the RP approach in detail. The algorithms governing the RP dynamically are presented and discussed, including some relevant implementation aspects. The performances and robustness of such algorithms are investigated through extended simulations, whose results are detailed presented. Finally, the relevant trade-offs and some possible refinements are illustrated.

1 Introduction

AQUILA is an international research project that is partly funded by EU (IST framework) [AQUILA]. The AQUILA project develops a Resource Control Layer (RCL) for DiffServ networks to provide a number of QoS Traffic Classes (TCLs) in addition to the traditional best effort.

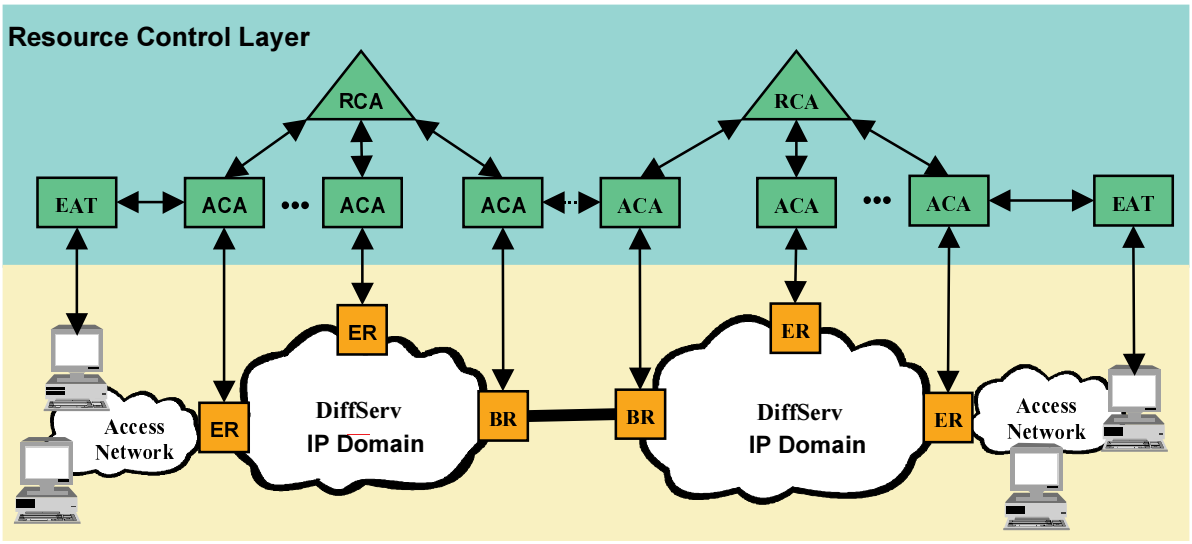

Figure 1: *The AQUILA Resource Control Layer (RCL).*

Four QoS TCLs have been defined in AQUILA (ref. to [TCL] for details). In an initial provisioning phase, the network operator determines the maximum amount of bandwidth that can be dedicated on each link to each TCL. It will be responsibility of the AC mechanism to ensure that the traffic injected in the network do not exceed such limits during normal network operation. Each Edge Router (ER) and Border Router (BR) have an associated Admission Control Agent (ACA) which handles AC requests. On the user side, the AC request for each QoS requesting flow is managed by the End-user Application Toolkit (EAT) on behalf of the applications. It is assumed that any application QoS requirement can be mapped into a Reservation Request (RR) for some amount of bandwidth (say B) for some TCL (say X). The mapping between the application QoS requirement and the RR pair <X,B> is done partially at the EAT (which chooses the relevant TCL, X) and partially in the ACA (which computes the effective bandwidth B from the declared traffic parameters). This way the only relevant resource for each TCL is *bandwidth*.

To gain scalability, in the AQUILA approach the AC decisions are processed locally at the ACA with no interactions with other elements. To accomplish that, it is needed to maintain information in the ACA about the maximum amount of traffic injectable in the ER for each TCL, called AC Rate Limit. The set of AC Rate Limits for the whole network is computed off-line in the initial provisioning phase, according to some optimal criteria depending on the expected spatial distribution of the offered traffic, with assigned constraints on the available link bandwidth for each TCL. Initial provisioning of the AC Rate Limit is performed by a centralised element, the Resource Control Agent (RCA), which is in charge of global resource management for the whole network, and which can communicate directly with the ACAs. The RCA, the ACAs and the EATs are the constituent element of the distributed RCL, as represented in figure 1.

As the optimal setting of the AC Rate Limits depends on the offered traffic distribution, it would be desirable to adapt such setting to track fluctuation in the offered traffic to some extent. Two mechanisms are considered in AQUILA to accomplish that: global re-provisioning and Resource Pools (RP). Global re-provisioning, i.e a new computation of the complete set of AC Rate Limits by the RCA, can be triggered by deep modifications in the long-term offered traffic. RP is an original mechanism used in AQUILA to track modifications in the offered traffic at a shorter time scale. The core idea is to organise adjacent ERs which share a common bottleneck into groups (called Resource Pool, RP) that can dynamically exchange their resource, i.e. co-ordinately increase / de-

crease their AC Rate Limits with no intervention of the centralised RCA. The RP solution preserves scalability and implementation simplicity.
The remaining part of this document is organised as follows. Section 2 describes the RP concept. Section 3 presents the basic resource management algorithm (RMA) used in the RPs, based on two watermarks on the used bandwidth. Section 4 presents extended simulation results for exemplary scenarios, covering huge parts of the parameter space. The results show that the proposed algorithm works properly. Nevertheless, it is found that the setting of the lower watermark parameter is critical to the performances and should be tuned carefully depending on the offered traffic process. To accomplish that, and automatic watermark control procedure is added to the algorithm in section 5 along with simulation results. Finally, conclusions and further work are discussed in section 6.

2 Dynamic Resource Management with Resource Pools

In order to get good scalability, the ACA should be able to process the RRs locally, without interacting with other network elements. To accomplish that we adopt an TCL-based provisioning scheme: for each TCL *j* each ER *i* is assigned a fixed amount of bandwidth $l_i(j)$, called AC Rate Limit. The ACA responsible for ER *i* can accept incoming reservation requests for TCL *j* until the sum of the bandwidth demands for the accepted flows remains below the limit $l_i(j)$. The value of the AC Rate Limit as well as the bandwidth demand for each active reservation are logically stored at the ACA. In order to gain a more dynamical behaviour, it would be desirable to adapt the values of the AC Rate Limits to the actual offered traffic, while respecting the assigned constraints on the link bandwidth sharing. In the present work we introduce the concept of Resource Pool (RP) to allow for some degree of dynamical sharing of resources between ER. The concept of RP arises from the general consideration that there are constraints between the AC Rate Limits of different ERs due to the topology and to the traffic distribution, and related to the presence of *bottlenecks*. A RP identifies a closed set of ERs sharing a common bottleneck and that can exchange resources with each other. Application of the RP concept is straightforward in the case a set of ERs are connected in a star to a Core Router (CR), attached to the bottleneck. The case is depicted in figure 2 (left), which will be used to illustrate the RP mechanism. The link rates indicated in the figure represent the maximum bandwidth allocable to the generic TCL *j* on each link. Assume that the value of 18 Mb/s between node 4 (the RP root) and the rest of the network represents the bottleneck which limits the total amount that can be injected by ERs 1 to 3 (the RP leaves) into the core. In the initial provisioning phase the AC Limits are set as follows: $l_1(j) = l_2(j) = l_3(j) = 5$ Mb/s, so that no more than 15 Mb/s can be injected by the ERs into the core. The RP root maintains a counter $f(j)$ for TCL *j* indicating the amount of available resource not allocated to the leaves, that is the difference between the maximum injectable traffic and the sum of AC Limits currently assigned to its leaves in the RP ($f(j) = 18 – 15 = 3$ Mb/s in the sample case). Whenever during network operation a RP leaf (say *i*) approaches the saturation of its resource budget $l_i(j)$ due to an increase in the offered traffic, it can ask the RP root for more resources (say Δ): if the request is accepted, the counters $l_i(j)$ and $f(j)$ will be respectively increased and decreased by Δ, which represent a shift of resources from the root to the leaf. Conversely, when the traffic offered to the leaf falls rather below its budget, the leaf itself can release an amount of resources to the root, decreasing the counter $l_i(j)$. Such resource will be eventually redistributed by the RP root to other requesting leaves, thus achieving resources shifting between the leaves. Of course the resources sharing is subject to some additional constrains: e.g. in the case of figure 2 each RP leaf can not be assigned an AC Limit larger than 10 Mb/s, which is the maximum rate assigned to TCL *j* on the link attached to the RP leaves.
Note that any resource request / release represent a transaction between a leaf and the root of the RP, and thus a message overhead. In general by tuning the algorithm parameters one can tune the frequency of such transactions, so to achieve a good compromise in the trade-off between message overhead and resources utilisation.

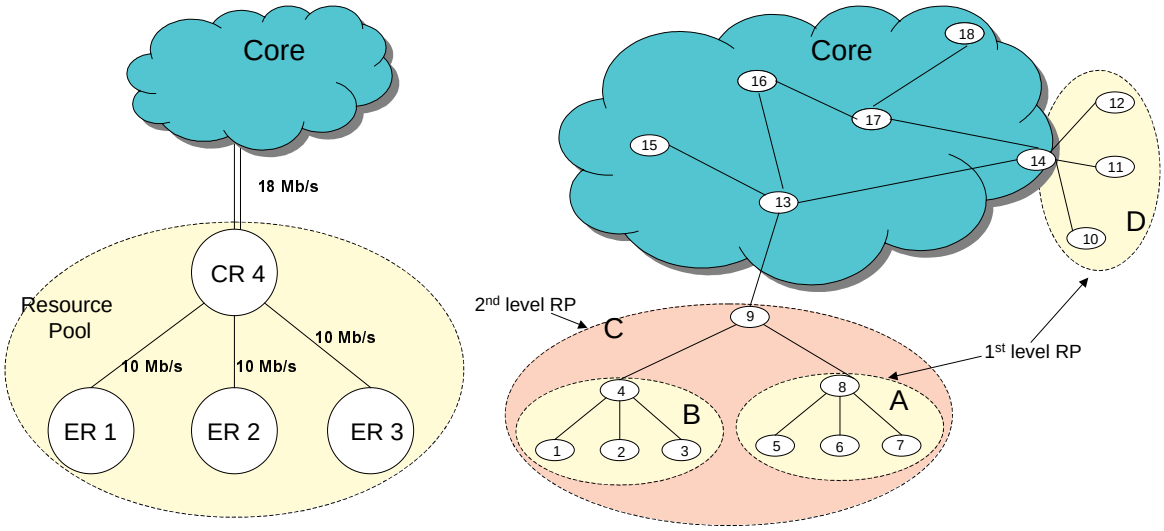

Figure 2: *Example of a Resource Pool (left) and Resource Pool Tree (right).*

The above approach can be hierarchically extended to build RP whose elements are not ERs but other RPs. The case is depicted in figure 2 (right), where RP "C" is composed by node 9 as root and RPs "A" and "B" as leaves. The hierarchical RP approach (also referred to as Resource Pool Tree, RPT) can be straightforwardly applied in those networks whose external areas present a hierarchically structured topology, which is expected to be a quite common case in practice.

3 The Basic Resource Management Algorithm

This section presents the Resource Management Algorithm (RMAs) used in the RP to exchange resources between the root and the leaves, hereafter called RP Elements (RPEs). Remember that each RPE can be a ER or a lower level RP. Different algorithms were developed and investigated in the framework of AQUILA. The RMA presented here is based on watermarks on the amount of locally used resources (bandwidth): RPEs use a high and a low watermark to exchange resources with its RP. Resource share adaptation actions are triggered, when the measured total resource demand leaves the resource window enclosed between the two watermarks. A RPE sends a resource request to its RP, when total resource demand exceeds the high watermark. A RPE returns unused resources, when total resource demand falls below the low watermark. This is the basic resource management scheme, which is very close to the RMA that is implemented in the AQUILA RCL currently and is described in [NPSV01]. Complete RMA operation is described later in section 5. A watermark adaptation procedure is added there.

A change in the resource distribution is always initiated by a leaf of a Resource Pool Tree (RPT), when an AC request arrives that cannot be accepted because of lack of resources. This can cause a chain of resource requests running up a RPT. All elements of this chain but the first are RPs. All RPs of this chain but maybe the last are RPEs of higher level RPs. In the following RMA description these recursive requests in a request chain running from RPs to RPs are left out for simplicity. RPs acting as RPEs just run the same RMA again, but resource adaptation activity slows down from the leaves to the root of a tree.

Notice that in view of a RPE there are two different resource requests: a primary request asking for resources of a RPE and possibly a triggered secondary resource request that is sent from the RPE to its RP in order to increase its resource share.

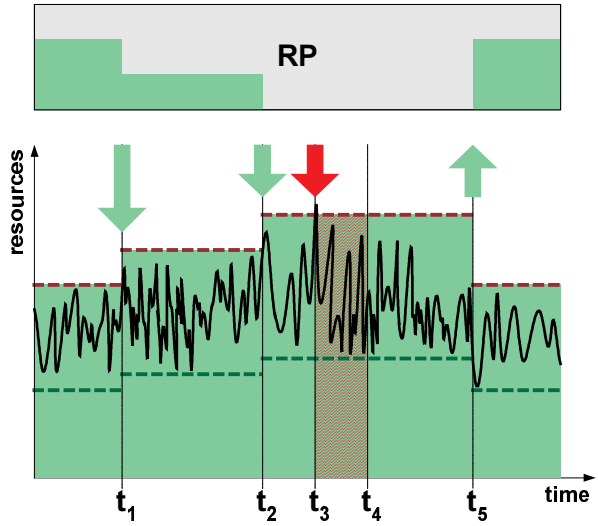

Figure 3: *Adaptation of the resource share of a RPE through the watermark RMA. Resource blocks are exchanged between the RPE (bottom)and its RP (top), when the resource demand (shaky black line) leaves the resource window between high watermark (dashed red line) and low watermark (dashed green line). Additional resources are requested from the RP, when resource demand hits high watermark. This happens at t_1, t_2 and t_3. Whereas additional resources are shifted from the RP to the RPE at t_1 and t_2, this is not possible at t_3 because the RP is empty at that time. So this request fails and further requests are inhibited until t_4. Unused resources are returned to the RP, when resource demand hits low watermark as at t_5.*

The resource adaptation process of the watermark RMA is depicted in figure 3. Let r be the size of the resource share that is assigned to a RPE. A high watermark $w_h \in [0,1]$ defines the threshold $w_h r$. A primary resource request is passed to the RP whenever this threshold is reached or crossed by the total resource demand. Before passing a primary resource request to a RP its resource demand is multiplied by n_{req}.

A low watermark $w_l \in [0,1]$ defines a second threshold $w_l r$. The fraction n_{rel} of the unused resources above the low watermark is returned to the RP whenever this threshold is reached or crossed by the total resource demand.
Altogether the RMA uses the following variables and parameters and works as follows:

Variables:

r	size of resource share that is assigned to considered RPE
u	amount of resources of RPE that is already reserved: $0 \le u \le r$
q	resource demand of considered primary resource request
t	actual time

Parameters:

$w_h \in [0,1]$	high watermark expressed as fraction of the assigned resource share r
$n_{req} = 1,2,3,...$	multiplier for primary demand q of resource requests that is used by the RPE when it forwards a resource request that cannot be satisfied to its RP
d_{req}	interval in seconds during that a RPE will not send further resource requests to its RP after a request was rejected
$w_l \in [0,1]$	low watermark expressed as fraction of the assigned resource share r
$n_{rel} \in [0,1]$	fraction of unused resources above low watermark that is returned when low watermark is crossed

RMA actions:
For each primary resource request demanding q resource units check if total demand exceeds the high watermark: if $u + q \geq w_h r$ then request an additional block of resources of size of $q \cdot n_{req}$ from RP. If a resource request is rejected by the RP then do not issue further requests before $t + d_{req}$.

For each primary resource release check if total demand falls below the low watermark: if $u \leq w_l r$ then return a block of unused resources of size of $(1 - w_l) r \cdot n_{rel}$ to RP.

4 Resource Pool Performance

The performance of RPs using the watermark RMA was investigated with a simulation model that was developed for the ns-2 from Berkeley [ns2]. Two performance investigations are presented here. First it is shown that a RP can adapt resource shares and balances oscillating load well using an exemplary scenario with three RPEs. Then RP performance is investigated systematically with a large number of simulations covering almost the total parameter space with an exemplary scenario of two RPEs and stationary load.

In the first scenario three RPEs share a common RP, see figure 4. A traffic generator is connected to each RPE that generates primary resource requests with exponential distributed inter-arrival times and exponential distributed holding times. Each RPE runs in addition to the RMA an AC function in this model. Requests are admitted as long as they require not more resources than are free in the resource share of the requested RPE at that moment. RPEs run the watermark RMA to adapt their resource shares to real resource demand. RPE 3 is offered stationary load with 100 flows on average. RPE 1 and 2 are offered oscillating loads as depicted in figure 5. Offered load oscillates with a 4 hour period between 20 and 45 flows on average. Load offered to RPE 1 is increased from 20 to 45 flows on average during the first hour, then stationary for one hour, decreased to the start level again in next hour and stays there for one hour until the whole cycle starts again. Load offered to RPE 2 is stationary for one hour first until the same oscillation starts. So it is phase shifted by one hour, see figure 6 which shows the offered load as well as the resource adaptation process. As can be seen resources are exchanged between RPE 1 and 2 continuously following load oscillations well. Figure 7 gives a more detailed view of the behaviour of the resource adaptation process. Resource shares follow resource demands at a coarser time scale wrapping up erratic demands with a

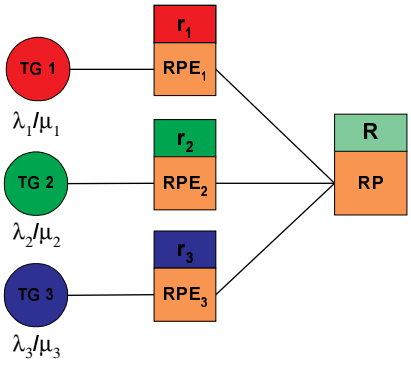

Figure 4: *Simulation model. Three traffic generators (TG 1 to 3) send reservation requests to three RPEs utilizing their resource shares r_1, r_2, and r_3 of the common resource pool R.*

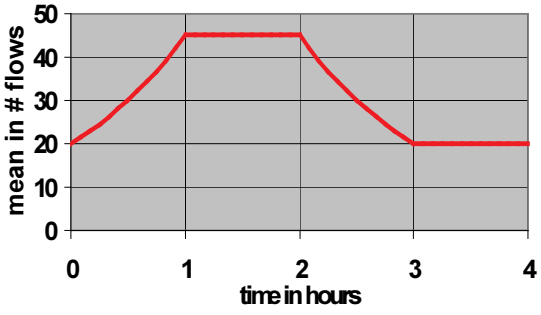

Figure 5: *Load oscillation pattern. With a timing of one hour each mean load is increased from 20 to 45, stationary at 45, decreased to 25 again and stationary at 20.*

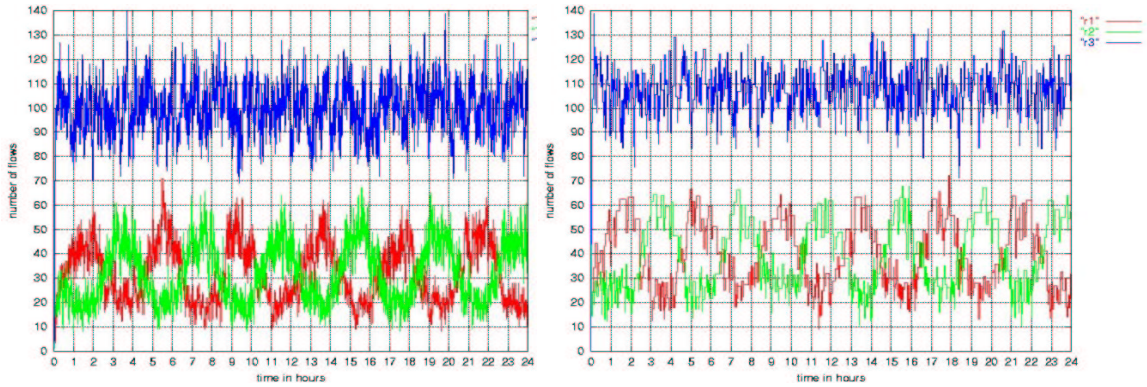

Figure 6: *Offered load (left) and resource adaptation process (right). Whereas stationary load was offered to RPE 3, RPE 1 and 2 had to cope with oscillating load. Resource assignments followed resource demand shifts very well.*

smoother step function.
Four performance measures were used to rate RMA performance: blocking frequency, fairness, adaptation rate, and capacity coefficient. Blocking frequency measures number of blocked primary resource requests accumulated over all RPEs in total. Individual blocking frequencies of the RPEs are used to measure fairness. The RMA is fair, if individual blocking probabilities equal global blocking probability and are all the same. Instead of the adaptation rate itself, the ratio of secondary to primary reservation activity was measured (number of secondary resource requests and releases measured at the RP divided by number of primary resource requests and releases measured at all RPEs in total). The capacity coefficient is the ratio of the pool size to the amount of resources needed to get the same blocking frequency with static resource assignments to all RPEs.
RP performance depends on resource provisioning policy of course, because available resources determine blocking probability. A rough target blocking probability of 1% was chosen. 210 resource units were given to the common pool with all RPE shares set to zero as the starting point. This are 10% less resources than are needed to meet the target blocking probability of 1% with a static resource assignments to the AC functions of the RPEs according to the Erlang-B formula.
The RP worked well with the parameter setting of table 1 in respect of all performance measures, see table 2. These results show that the RP was able to shift resources between RPEs following changing demands with a resource shift frequency 2 orders below the primary resource request rate.

For a systematic performance investigation covering almost the whole parameter space two RPEs sharing a common pool were used as shown in figure 8. Static load with a mean of 20 flows resp.

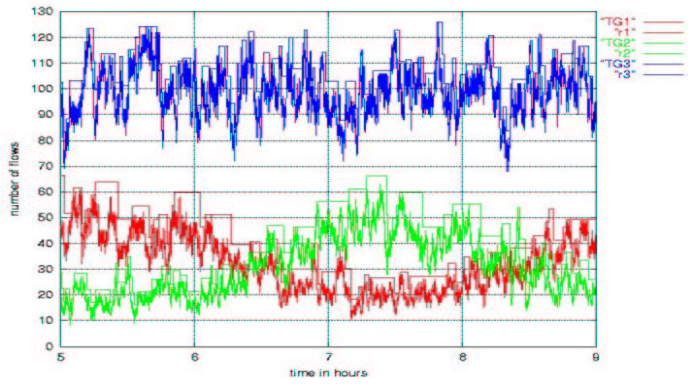

Figure 7: *Detailed view of the resource adaptation process of the watermark RMA. Resource assignments take the form of a step function wrapping up restless bouncing demands closely.*

Parameter	*RPE 1*	*RPE 2*	*RPE 3*
w_h	1.0	1.0	1.0
n_{req}	5	5	5
d_{req}	60	60	60
w_l	0.55	0.55	0.80
n_{rel}	0.50	0.50	0.50

Table 1: *Parameter setting.*

capacity coefficient	γ = 0.86
blocking frequency	b = 0.4%
fairness	b_1 = 0.3% b_2 = 0.2% b_3 = 0.4%
activity ratio	a = 0.9%

Table 2: *Performance measurement results.*

100 flows was offered to RPE 1 resp. RPE 2. The RP started with 147 resource units in the common pool and 0 resources at the RPEs. This is the amount of resources needed to meet 1% blocking probability with fix resource assignments. Figure 9, 10, and 11 show the result.
The low watermark determines willingness to share resources. Willingness to share decreases with lower low watermark values. Lower low watermarks are crossed less frequently thus freezing bonded resources. This has a large impact on performance as can be seen in figure 9. There is a trade-off between RP activity and its performance. Best performance in view of a single RPE is obtained with high low watermark values at competing RPEs. But this leads to high low watermarks for all RPEs and high activity rates, see bottom picture of figure 9. Altogether there is a small area in the front right quadrant of the low watermark plane only which results in fair resource sharing and low activity rates. It is the task of the watermark adaptation scheme that is presented in next section to keep RPEs in this area.
The other RMA parameters are much less sensitive. Release block size has almost no influence as shown in figure 10. Impact of request block size is larger but still very small compared to low watermark, see figure 11.
Variation of high watermark w_h and request delay d_{req} are not shown here because of lack of space. If high watermark is set to 1.0, additional resources will be requested if actually needed only. With high watermarks $w_h < 1.0$ resource shares are increased in advance. This is an advantage if processing delay for secondary resource requests is high. Otherwise high watermarks should be as high as possible, because lower high watermarks increase activity.
Inhibiting secondary resource requests for 60 seconds after a request was rejected, which was used in figure 9 to 11, roughly halves activity rate. The setting of this parameter is not critical because it has an exponential impact on the activity rate. Request delays greater than zero decrease activity rate quickly first then less and less. And impairments of blocking probability show up very slowly only.

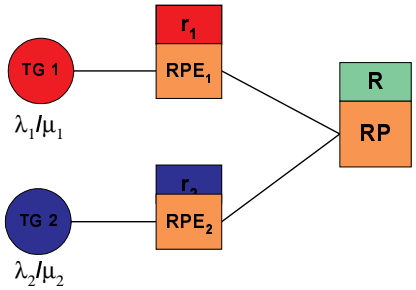

Figure 8: *Simulation model. Two traffic generators send reservation requests to two RPEs utilizing their resource shares r_1 and r_2 of the common resource pool R.*

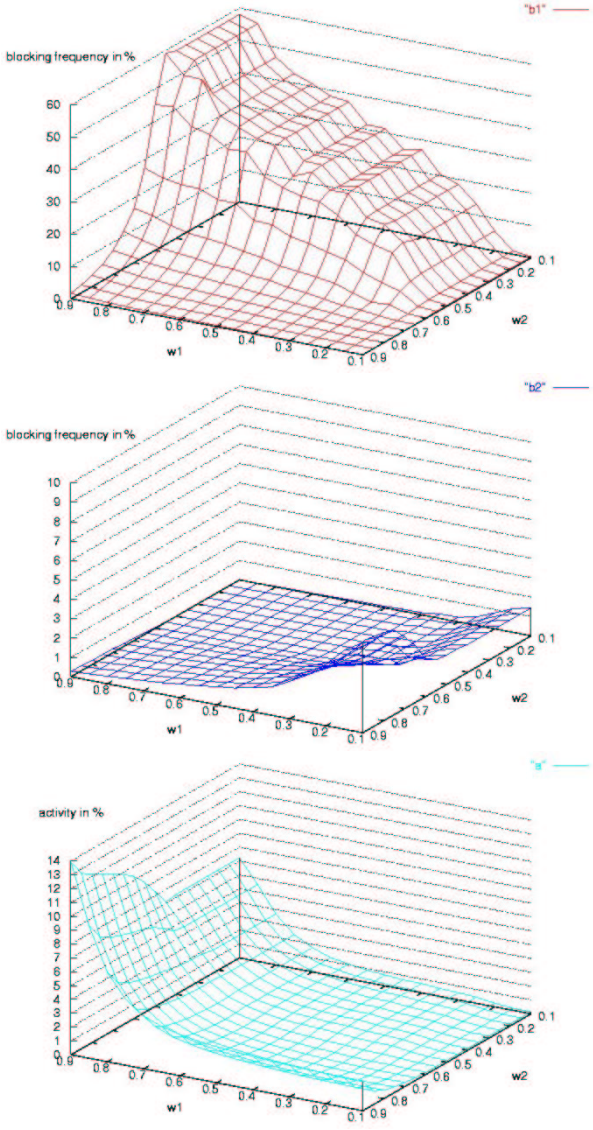

Figure 9: *Measured impact of low watermark (**w_1, w_2**) on blocking frequency of RPE 1 (top) and RPE 2 (middle) as well as on resource re-distribution activity (bottom).*
The top picture shows severe performance problems at RPE 1, when small values are used for the low watermark of RPE 2 ($w_2 \leq 0.6$).
The picture in the middle shows similar but less severe performance problems at RPE 2 when small values are used for the low watermark of RPE 1 ($w_1 \leq 0.3$).
Finally the bottom picture shows increase of resource adaptation rates, when large values are taken for low watermarks.

Parameter setting:
$w_h = 1.0$
$d_{req} = 60$ sec
$n_{req} = 5$
w_l : $\mathbf{w_1, w_2 \in [0.10, 0.90]}$
$n_{rel} = 0.50$

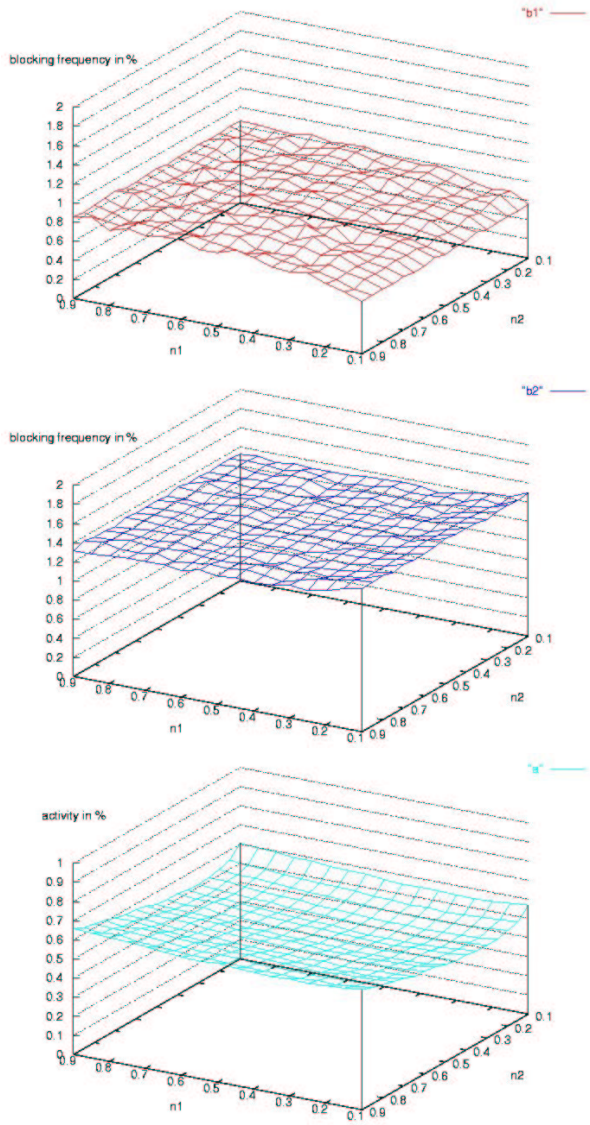

Figure 10: *Measured impact of release block size (**n_1, n_2**) on blocking frequency of RPE 1 (top) and RPE 2 (middle) as well as on resource re-distribution activity (bottom).*
Performance is insensitive to this parameter.

Parameter setting:
w_h = 1.0
d_{req} = 60 sec
n_{req} = 5
w_{l1} = 0.33
w_{l2} = 0.75
*n_{rel}: **n_1, n_2 ∈ [0.10, 0.90]***

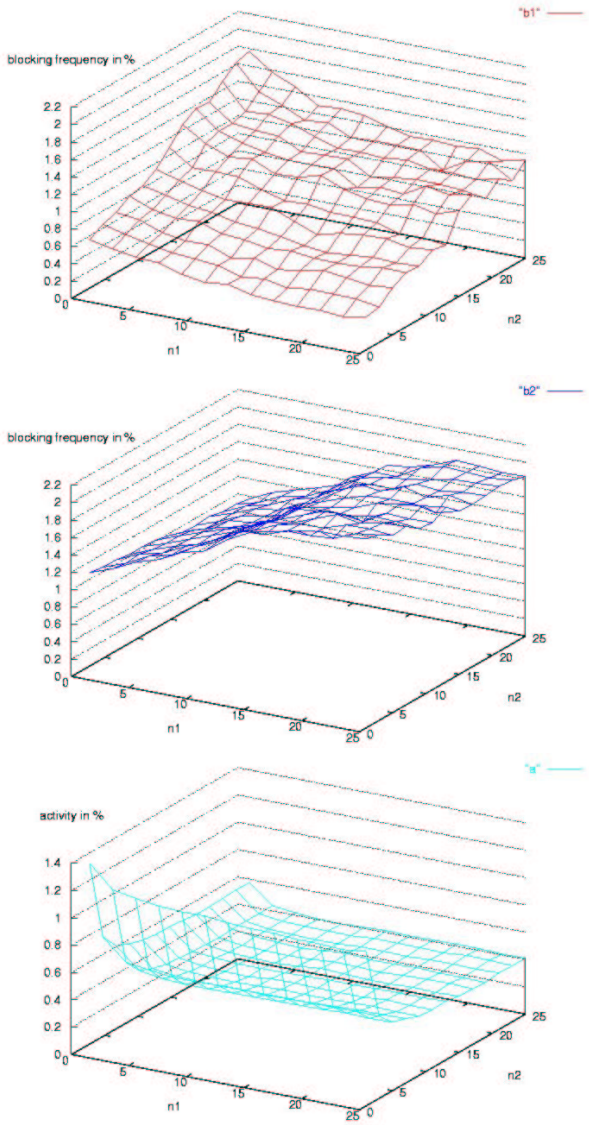

Figure 11: *Measured impact of request block size (**n_1, n_2**) on blocking frequency of RPE 1 (top) and RPE 2 (middle) as well as on resource re-distribution activity (bottom).*
This parameter has a small influence on performance only.

Parameter setting:
w_h = 1.0
d_{req} = 60 sec
*n_{req}: **n_1, n_2 ∈ [1, 25]***
w_{l1} = 0.33
w_{l2} = 0.75
n_{rel} = 0.50

5 The Full RMA Including A Watermark Adaptation Scheme

The pervious performance analysis showed that the RMA is able to work well but that there is a rather small area in its parameter space yielding good results. The low watermark is a problem, because it highly depends on unknown traffic load. Our solution is to let the RMA not only adapt the resource share of its RPE but to let it adapt its low watermark too. Therefore it measures the ratio of its secondary to primary resource return rate and will decrease or increase the low watermark, if estimated return rate is too high respective too low.

Figure 12 and 13 show the watermark adaptation process using a very rough first adaptation scheme. If no resources were returned after 1000 primary resource releases, return rate was assumed to be too low. If resources were returned 5 times to the RP during less than 1000 primary resource releases, return rate was assumed to be too high. If resources were returned 1 to 4 times during 1000 primary resource releases, return rate was assumed to be ok.

If estimated return rate is too high, low watermark is decreased: $w_l \rightarrow w_l(1-\alpha)$

If estimated return rate is too low, low watermark is increased: $w_l \rightarrow w_l(1+\beta)$

Simulations of section 4 were repeated using the complete RMA with increments and decrements of 5% ($\alpha = \beta = \mathbf{0.05}$) for a first analysis. The RMA had to find the right low watermark values itself this time. Figure 12 shows the watermark adaptation process for stationary load offered to the RPEs in the scenario of figure 8. Figure 13 shows the watermark adaptation process for oscillating load offered to the RPEs in the scenario of figure 4. Low watermarks converges in both cases to useful values that yield good performance measures.

Further adaptation schemes will be investigated, like additive increments and decrements or adaptive increments and decrements using measured deviations of return rate form its target value or an estimated trend observed through a series of measured deviations.

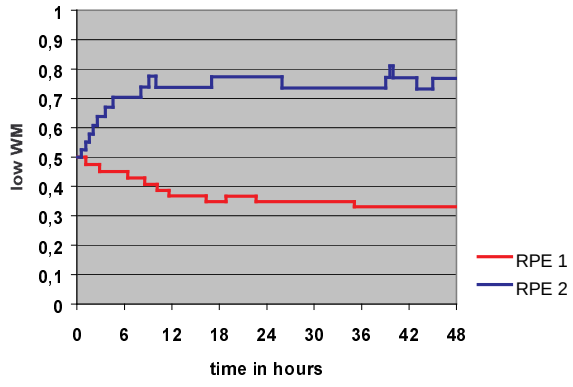

Figure 12: *Watermark adaptation process using scenario of figure 8 and a resource pool size of 147 resource units.*
Measured performance after first hour:
capacity coefficient: γ= 1.01
blocking frequency: b = 1.3%
fairness: b1 = 1.6%, b2 = 1.2%
activity ratio: a = 0.6%

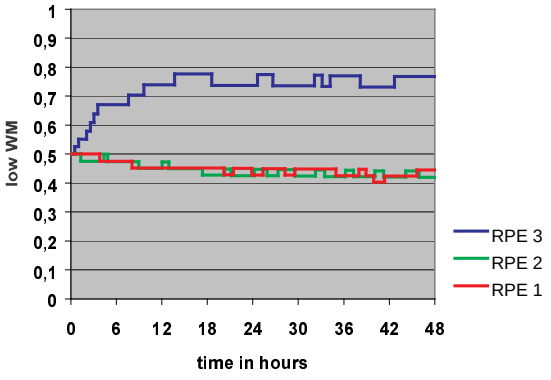

Figure 13: *Watermark adaptation process using scenario of figure 4 and a pool size of 210 resource units.*
Measured performance after first hour:
capacity coefficient: γ= 0.89
blocking frequency: b = 0.8%
fairness: b1 = 0.6%, b2 = 0.7%, b3 = 0.8%
activity ratio: a = 0.5%

6 Conclusions

Resource pools are used in the AQUILA RCL for online adaptations of the resource provisioning to AC. Groups of Resource Pool Elements share common pools of resources in a hierarchical manner. RPEs measure real resource demand continuously during network operation and shift unused resources to demanding locations. This compensates resource provisioning mismatches due to forecast errors and fluctuating traffic demands like temporary hot spots attracting large traffic volumes.
The watermark resource management algorithm for resource pools was presented. Simulations showed that resource pools perform well: they achieve blocking probability targets with reasonable resources and distribute resources in a fair manner. Also, we showed that resource adaptation activity, which is the frequency of triggered adaptation actions, is 2 orders of magnitude below the primary resource reservation activity at RPEs, i.e. the frequency of AC requests at the ERs.
The concept of resource pools will be improved during our ongoing AQUILA project. Next steps are refinements of the watermark adaptation scheme, analysis of hierarchical RP systems and performance studies using different traffic models.

7 References

[AQUILA] AQUILA home page: http://www-st.inf.tu-dresden.de/AQUILA

[NPSV01] Eugenia Nikolouzou, George Politis, Petros Sampatakos, Iakovos Venieris: An Adaptive Algorithm for Resource Management in a Differentiated Services Network, ICC 2001, Helsinki, Finland, 11 - 15 June 2001

[ns2] network simulator version 2, http://www.isi.edu/nsnam/ns/

[TCL] G. Eichler, R. Widera, S.Salsano, F.Ricciato, A.Thomas: AQUILA Network Services, COMCON 8, Crete, Greece, 25 - 29 June 2001