An Adaptive Algorithm for Resource Management in a Differentiated Services Network

E. Nikolouzou, G. Politis, P. Sampatakos, I. S. Venieris
National Technical University of Athens, Greece
National Technical University of Athens,
Department of Electrical and Computer Engineering,
9 Heroon Polytechniou str, 157 73, Athens, Greece
Telephone: +30 1 772 2551, FAX: +30 1 772 2534
E-mail: {enik, gpol, psampa}@telecom.ntua.gr, ivenieri@cc.ece.ntua.gr

***Abstract*-Internet is widely known for lacking any kind of mechanism for the provisioning of Quality of Service guarantees. An overlay Resource Control Management Layer on top of a Differentiated Services core network is introduced for managing and adjusting the resources among network elements. This layer realises an algorithm which provides a dynamic approach for resource distribution. Our experimental results show that this algorithm can allocate network resources according to traffic load and provide an adaptive and efficient way for re-distributing the resources among metwork elements.**

***Keywords*-Quality of Service, Bandwidth Brokers, Differentiated Services, Admission Control, Resource Distribution, Resource Control Management Layer**

I. INTRODUCTION

The enormous rise in usage and popularity of Internet as well as the introduction of new applications, such as voice, video and advanced multimedia applications, have motivated the Internet Community towards the research for improving the Quality of Service (QoS) provided by today's best effort networks.

The Differentiated Service (DiffServ) model [1,2] has become a preferred solution, which provides a scalable means for supplying multiple levels of service, based on handling of traffic aggregates. This architecture achieves scalability by maintaining simple functionality at the core network and by shifting complex mechanisms only at the edges of the network.

Nevertheless, the DiffServ architecture does not specify any kind of mechanism for an overall resource management and admission control. The Internet2 project [3] proposes the Bandwidth Broker (BB) [4] architecture that controls the resources of a domain, among others. The proposed network architecture described in this paper is based on the DiffServ model and the Bandwidth Broker concept. Its objective is to enhance the original DiffServ architecture by adding a new layer, the Resource Control Management Layer (RCML) above it in order to provision QoS features to the customers of the network. The RCML is basically a realisation of a distributed BB architecture, promising scalability and efficiency characteristics.

The RCML is composed of different distributed entities organised in a hierarchical manner, each one managing the resources assigned to it. The algorithm that assigns the initial resources to these entities and further reassigns them according to resource reservation requests is the main focus of this paper.

The paper is organised as follows: Section 2 describes the overall architecture, Section 3 proposes an algorithm for resource control and distribution. Finally, Section 4 gives an estimation and evaluation of the parameters of the proposed algorithm.

II. ARCHITECTURE

In this section the overall architecture will be described and the RCML will be particularly examined.

A. Architectural Principles

The proposed architecture aims to provide an efficient way for managing the resources available to the network. It consists of two functional areas: the data plane that is responsible for transmitting Internet Protocol (IP) packets and an overlay control plane, namely the Resource Control Management Layer (RCML). The RCML consists of three logical entities, as depicted in Fig. 1:

- The Resource Manager Agent (RMA) that is the highest authority in an administrative domain. It is responsible for admission control decisions and management of the network resources. Moreover, it has the overall view of the policies enforced in a domain, and decides for the router configuration and management of the bilateral Service Level Agreements (SLA) between adjacent administrative domains.
- The Access Control Agent (ACA) that basically controls the user access to the network by performing policy control, as well as authorisation and accounting functions. Moreover, each ACA is assigned the task of controlling an Edge Device (ED) i.e. configures the appropriate ED parameters after a reservation request is admitted by the RCML.
- The End-User Application Toolkit (EAT) that provides an interface to the end-user applications that enables them to signal their requirements to the QoS infrastructure.

The configuration described in the paper considers only one RMA per Internet Service Provider (ISP). In addition in this paper only intra-domain issues are examined, while inter-domain aspects are under research and they will be addressed in a future work.

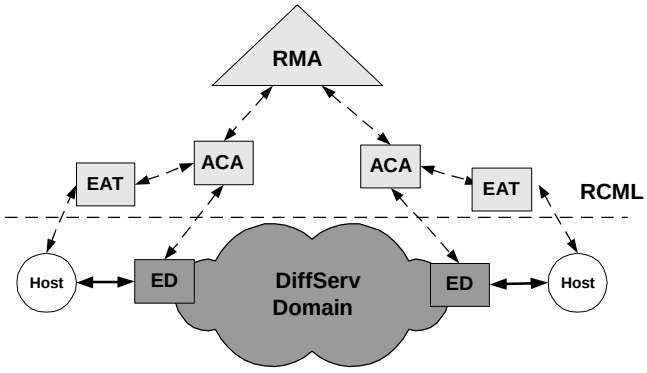

Fig. 1: Resource Control Management Layer Architecture

B. *Hierarchical Structure of the RMA*

In order to simplify the task of the RMA to handle the network resources efficiently, the network is divided into sub-areas that form a tree structure, where each sub-area is assigned its own resources. The network administrator estimates these resources according to traffic load forecasts and/or results retrieved by a measurement-based platform.

It has been examined that lately the number of user requests is dramatically increasing and a standalone management entity could not perform well under these conditions.

Therefore, the RMA is divided in logical entities (Resource Managers, RM) and each one of them is assigned the task of managing the resources of a sub-area. The RCA is based on the hierarchical structure depicted on Fig. 2.

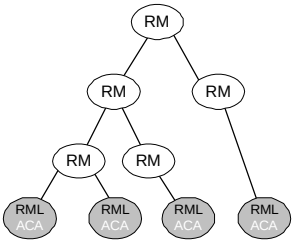

Fig. 2: Hierarchical Structure

Every node of the tree has none or many children and exactly one father, except from the root node that has no father. In addition, each leaf of the tree structure (Resource Manager Leaf, RML) is associated to one Access Control Agent (ACA). During the start-up configuration procedure, the RMs/RMLs are assigned their initial resources, which are provided by a database managed by the network administrator. These initial resources may not reflect the actual traffic load of each sub-area, therefore, the RMs/RMLs should be able to adjust resource assignments to real traffic conditions, which are difficult to be forecasted and may change during time.

Since the RMs/RMLs are distributed and need to communicate for the re-assignment of the resources among them, the CORBA technology [5] is adopted for the RMA implementation.

In order to keep the interactions between two nodes as simple as possible, an event-driven model is adopted, where a child RM/RML always requests more resources from its father or releases any unused ones. In this way the father is continuously aware of his current available resources in order to further distribute them as efficiency as possible. Consistently it is provided to the father RM a kind of dynamic and automatic updating of the current status of his resource allocations. In this way the implementation complexity is kept really low, without the burden of a reverse interaction.

The algorithm that decides when a child should ask for more resources from its father or give back the unused ones as well as the calculation of the corresponding amount of resources is described in the next section.

III. DESCRIPTION OF THE ALGORITHM

In this section an algorithm for resource distribution and redistribution will be presented.

A. *Algorithm mechanism*

The basic mechanism of the algorithm is to handle efficiently the cases when re-distribution of resources is needed.

This is invoked when an RML does not have enough resources to accommodate a new user request. According to the algorithm realised, an RML will make a request for additional resources to its father. The child makes a request determining the minimum additional resources needed to admit the request and an upper limit for the resources that can accept from its father.

The father is responsible for deciding how many resources to give to its child, depending on the amount of resources requested, the upper limit defined by the child and the amount of its free resources. In case, the father does not have enough resources will also make a resource request to its father RM (of the above level). This procedure can continue up to the root of the tree. The procedure of finding additional resources is bottom-up, i.e. from the leaves of the tree up to the root.

B. *Initial Resource Distribution*

The network administrator is responsible for defining the initial resources to be distributed to the nodes of the tree. Each RM distributes its resources to its children according to the initial amounts defined. After this top-down start-up procedure, initial resources are assigned to all nodes of the tree.

Each RM and RML is basically described by the following set of parameters:

R_{max} *: upper limit of resources that can be assigned to an RM/RML*

R_{tot} *: current resource assignment to an RM/RML*

R_{res} *: current reserved resources of an RM/RML*

R_{free} *: currently unused (free) resources of an RM/RML*

R_{av} *: maximum resources that can be additionally assigned to an RM/RML*

The R_{max} defines an upper limit for the traffic that an RM/RML can afford.

The equations (1)-(6) describe the initial resource status of an RM/RML as well as the relation of the resources of a father RM and its children (f: father, c: children):

$$R_{max} \geq R_{tot} \geq 0 \quad (1)$$

$$R_{free} = R_{tot} - R_{res} \quad (2)$$

$$R_{av} = R_{max} - R_{tot} \quad (3)$$

$$R^{f}_{res} = SR^{c}_{tot} \quad (4)$$

$$R^{f}_{max} \geq R^{c}_{max} \quad (5)$$

$$SR^{c}_{max} \geq R^{f}_{max} \quad (6)$$

C. *Resource Distribution*

After the initialisation of the tree and the assignment of the initial set of resources, user makes its resource reservation requests to the EAT, which forwards these requests to the ACA. Under the condition that the user access to the network is verified, ACA hands over this request to the corresponding RML for admission control.

A number of additional parameters must be defined first. Fig. 3 depicts those parameters giving a more comprehensive view of the implemented algorithm.:

R_{req} *: minimum resources requested from an RM/RML*

R_{recv} *: resources actually received from a child after a request for more resources to its father*

A_{max} *: number of max resource shifts; father RM increases the resources of its child by* $A_{max} \cdot R_{req}$

A_{min} *: number of min resource shifts; father RM increases the resources of its child by* $A_{min} \cdot R_{req}$

A_{med} *: number of resources shifts; father RM increases the resources of its child by* $A_{med} \cdot R_{req}$ $(A_{max} < A_{med} < A_{min} < 1)$

w_L *: a low limit for the free resources of the RM*

w_H *: a high limit for the free resources of the RM* $(w_L < w_H < 1)$

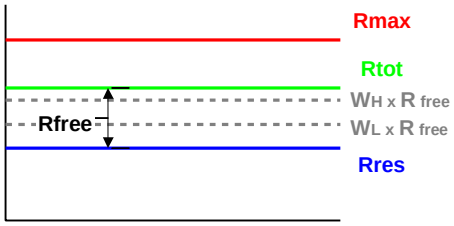

Fig. 3: Parameters of the Resource Distribution algorithm

As long as the RML has enough resources to accept a reservation request, there is no need of redistribution of the resources. In case an RML does not have efficient resources to accommodate an R_{req} for a reservation it asks more resources from its father RM, and the latter decides how much to give back to it, R_{recv}. The same procedure can be repeated many times, up to the root of the tree.
The steps of the proposed algorithm executed by the RML after a resource reservation request are:

1. ***if*** $R^{RML}_{res} + R_{req} > R^{RML}_{max}$
   then reject *the request;*
2. ***if*** $R^{RML}_{res} + R_{req} <= R^{RML}_{tot}$
   then *admit request*
   $R^{RML}_{res} = R^{RML}_{res} + R_{req}$
   end then (2)
3. ***else if*** $R^{RML}_{res} + R_{req} > R^{RML}_{tot}$
   then *calculate resources to ask from father*
   $x = (R^{RML}_{res} + R_{req}) - R^{RML}_{tot}$
   make a request to father:
   $R^{RML}_{av} = R^{RML}_{max} - R^{RML}_{tot}$
   $R_{recv} = request(x , R^{RML}_{av})$;
   if *request accepted by father RM*
   then *admit the request*
   change total and reserved resources:
   $R^{RML}_{tot} = R^{RML}_{tot} + R_{recv}$
   $R^{RML}_{res} = R^{RML}_{res} + R_{req}$
   end then
   else *reject the request;*
   end then(3)

When a father receives a request for additional resources, *request(x , R_{av})* , it calculates the actual additional resources that can return to its child. The father will give a multiple ($A_{max}/A_{min}/A_{med}$) of the *x* depending on the amount of its free resources R_{free}. The upper limit of the resources that can be assigned to, is bounded by the R_{av} (R^c_{av} in request() below). In case a father RM can not assign to its child not even the minimum amount of resources requested, it calls the same request function to its corresponding father. The realisation of the request() is given:

1. ***if*** $A_{max} \cdot x <= w_L \cdot R_{free}$
   then $R_{recv} = min(A_{max} \cdot x, R^c_{av})$
   $R_{res} = R_{res} + R_{recv}$
   return R_{recv}
   end then (1)
2. ***else if*** $A_{med} \cdot x <= w_H \cdot R_{free}$
   then $R_{recv} = min(A_{med} \cdot x, R^c_{av})$
   $R_{res} = R_{res} + R_{recv}$
   return R_{recv}
   end then (2)
3. ***else if*** $A_{min} \cdot x <= R_{free}$
   then $R_{recv} = min(A_{min} \cdot x, R^c_{av})$
   $R_{res} = R_{res} + R_{recv}$
   return R_{recv}
   end then (3)
4. ***else*** *ask resources from its father*
   $x' = (R_{res} + x) - R_{tot}$
   $R_{recv} = request(x', R_{av})$
   if *request accepted by father*
   then
   $R_{tot} = R_{tot} + R'_{recv}$
   goto step(1)
   end then
   else *reject the request;*

end then (4);

The w_L and the w_H define two limits for the free resources of an RM. Depending on the ration of the reservation request to a limit of the free resources, an appropriate multiple of the requested resources is given. In case that a multiple of the requested resources is less than the low level of free resources, then the A_{max} of the requested resources is given. That implies that requested resources are really very small in comparison to the free resources of the RM.

D. Resources Release

When a user makes a release request to the RML, the latter deletes the reservation and checks whether or not it can release any unused resources to its father. In order to take such decision an additional set of variables are defined:

l	*: a low watermark, l<1*
R_{res}	*: reserved resources before the deletion of reservation*
R'_{res}	*: reserved resources after the deletion of reservation*
R_{rel}	*: resources to be released to the upper level*
a	*: a < 1*

The low watermark, *l*, is used to check the current status of reserved resources of an RM/RML. In case the reserved resources are below this watermark, this indicates that there are unused resources that should be given to the upper level. The amount of released resources should be calculated considering the trade-off between giving as much as possible and keeping resources for future use.
The algorithm for deciding and calculating the resources to be released is:

1. *After the deletion of reservation reserved resources are* R'_{res}
2. ***if*** *(* $R'_{res} < R_L$ *) and* ($R_{res} > R_L$)
   then
   calculate how much to release to the upper level
   The new reserved resources after the release must be between the R'_{tot} *and* $l \cdot R'_{tot}$*:*
   - $R'_{res} = a \cdot (R'_{tot} + l \cdot R'_{tot})$
   - $R'_{tot} = R_{tot} - R'_{rel}$

   From above: $R'_{rel} = R_{tot} - R_{res} / (a(1 + l)$
   release R'_{rel}
   end then (2)
3. ***else*** *do not relase resources.*

The value of *a*, determines indirectly the actual amount of resources released. In fact, it specifies the desired revel of reserved resources in the new state of total resources of an RM/RML (after an amount of resources has been released, i.e. R'_{rel}).

IV. SIMULATION

A. Simulation methodology

Simulations were carried out in a Pentium III PC with the help of a special tool that has been developed in JAVA programming language. In order to understand fully the behavior of the algorithm,

a tree structure has been defined and implemented, depicted in Fig. 4. The actual tree structure does not play a crucial role for the study of the proposed algorithm.

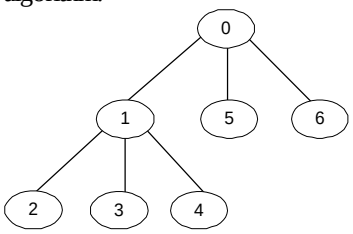

Fig. 4: Simulation topology

A simulation experiment consists of a random process of reservation request arrivals. Each request arriving to an RML may be admitted or rejected according to the specifics of the algorithm in question. The inter-arrival time of reservation request follows an exponential model, while the size of the resources requested follows a random model. A reservation request may arrive to each leaf node with the same probability. The reservation requests may have a capacity of 64, 128 or 512Kbps. More precisely, the capacity specifies the peak rate of reservation requests originated by the host. Other details that specify the nature of the requests e.g. average rate, are omitted since they are not necessary for the description of the objectives of the implemented algorithm.

During simulations the average utilization of each node of the tree and the number of interactions (request() calls) invoked are measured, using different algorithm configurations.

B. Results

A number of simulations have been carried out, where the behaviour of the algorithm has been examined under different set of values of parameters. TABLE I summarises those parameters and assigns to them a possible value.

TABLE I
MAIN VARIABLES OF THE ALGORITH

Variable	Value
A_{max}	5(3-8)
A_{med}	3(2-4)
A_{min}	1(1-3)
w_L	0,2
w_H	0,6
l	0,5-0,7
a	0,5

We have examined our algorithm under two basic cases: with initial resources, R_{tot} and with zero initial R_{tot} resources. The second condition can be preferred when there are not actual forecasts of the traffic load of the sub-areas of the network, so resources are distributed according to the demand. The root of the tree only has an initial R_{tot}.

Primarily we have examined the variation of R_{tot} and R_{res} in time for all the RMs/RMLs of the tree, changing the values of the parameters in TABLE I for both situations. In general the algorithm offers an exceptional adaptability as indicated in Fig. 4 for an RML. The adaptability of R_{tot} to the reserved resources, R_{res}, depends mainly on the values of A_{max} *and* l. The greater the value of A_{max} the less adaptive the algorithm becomes, since a greater amount of resources will be re-assigned to a child after a request() call. The value of l determines the level that resource release must be, meaning that the greater its value is, the sooner unused resources will be released to the upper level.

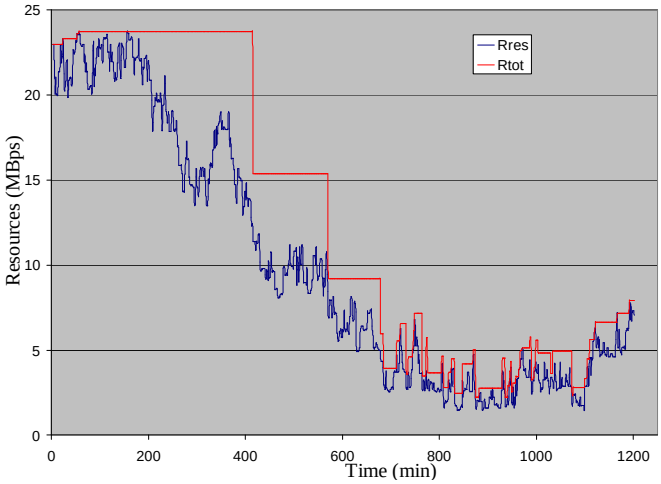

Fig. 5: Status of Resources of a RML

In sequence the number of interactions among all nodes of the tree was examined for different values of A_{max}, for both cases. As depicted in Fig. 5 the greater the value of A_{max} the smaller the number of interactions for both cases. The zero initial resources may provide generally an additional efficiency and adaptability, but in the beginning of resource reservation phase every new reservation request invokes a set of sequential request() calls from the RML up to the root of the tree. That justifies the large deviation of the number of interactions among the two cases.

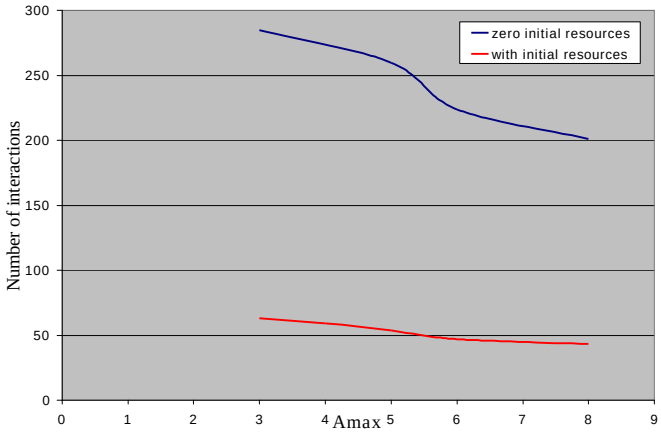

Fig. 6: Number of interactions in relation to Amax

Another crucial characteristic for the performance of the proposed algorithm is the utilisation of the network resources. The average utilisation has been measured for the both cases, varying the value of A_{max}, as illustrated in Fig. 6. The algorithm really provides a high utilisation, which is inversely proportional to the value of A_{max}.

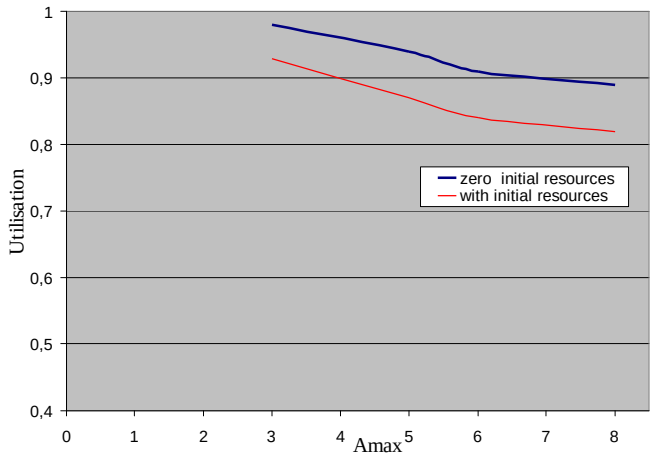

Fig. 7: Utilisation in relation to Amax

The current utilisation of resources of each node depends also directly on the value of *l*, since *l* composes an under bound for the utilisation.

It has been also examined the response of the algorithm to the modification of values of the other parameters. A_{min}, A_{med}, w_H and w_L also influence the utilisation and the number of interactions in the same way as A_{max}, but they have a smaller impact than A_{max}. In addition the behaviour of the parameter *a* is identical to that of *l*, since they both determine the state that release of resources should take place.

Summarising, there is trade-off between the utilisation of network resources and the interactions between the nodes of the tree. When the main goal of the implementation is a small number of interactions among the remote nodes for improving the performance, then a relatively large value of A_{max} is required. Consequently, a smaller utilisation of network resources is achieved. It depends also on the network administrator to tune appropriately the value of A_{max} and the other parameters in order to achieve the desired performance.

V. CONCLUSIONS

The overall architecture presented in this paper addresses the problem of QoS provisioning in IP networks, providing a distributed BB architecture. The RCML is introduced and is responsible for the handling of the reservation requests, performing policy-based admission control, configuring the network in a top-down approach, managing the network resources and dynamically redistributing them among the network elements.

This paper implements and evaluates the algorithm used by the RCML for distribution and re-distribution of resources of the underlying network. The proposed algorithm provides a high performance and a great adaptability, even in a highly random traffic model, while the network resources are used really efficiently.

ACKNOWLEDGMENT

This work was performed in the framework of IST Project AQUILA (Adaptive Resource Control for QoS Using an IP-based Layered Architecture - IST-1999-10077) [6] funded in part by the EU. The authors wish to express their gratitude to the other members of the Aquila Consortium for valuable discussions.

REFERENCES

[1] S. Blake et al., "An Architecture for Differentiated Services", RFC 2475

[2] K. Nichols, V. Jacobson and L. Zhang, "A Two-bit Differentiated Services Architecture for the Internet", RFC 2638

[3] Teitelbaum et al. "Internet2 QBone: Building a Testbed for Differentiated Services", *IEEE Network*, September/October 1999, Vol.13, No.5, pg. 8-16.

[4] Neilson, R.; Wheeler, J.; Reichmeyer, F.; Hares, S.: A Discussion of Bandwidth Broker Requirements for Internet2 Qbone Deployment, *Internet2 Qbone BB advisory Council*, August 1999.

[5] CORBA/IIOP2.3.1.Specification. http://www.omg.org/corba/cichpter.html

[6] Aquila project: http://www-st.inf.tu-dresden.de/aquila/