Evaluation of an Algorithm for Dynamic Resource Distribution in a Differentiated Services Network

E.G. Nikolouzou, P. D. Sampatakos, I. S. Venieris
National Technical University of Athens, Greece

National Technical University of Athens
School of Electrical and Computer Engineering
Telecommunications Laboratory
9 Heroon Polytechniou str, 15773 Athens, Greece
e-mail: {enik, psampa}@ telecom.ntua.gr,
ivenieri@cc.ece.ntua.gr

Abstract. New applications have been introduced to the today's "best-effort" IP networks having different bandwidth and delay guarantee requirements. The IETF is currently focused on Differentiated Services as the architecture to provide Quality of Service to IP networks. Towards this effort, an overlay Resource Control Layer on top of a Differentiated Services core network is introduced in this paper, in order to provide a simple control plane architecture that enables the overall handling of network resources and the configuration of network elements in a domain. Therefore, a dynamic algorithm is proposed for that layer to manage, adjust and distribute resources in an efficient and dynamical way. The simulation results show that this algorithm provides a significant improvement in bandwidth assurance and utilization of network resources compared with a static resource assignment approach, keeping at the same time complexity at a low level.

INTRODUCTION

The Internet today provides a best-effort architecture, which is basically ideal for elastic applications, such as e-mail and file transfer. The network traffic though has increased as the number of users and applications has also increased. Moreover, the Internet traffic has also changed in character; new bandwidth-demanding and delay-sensitive applications (voice-over-IP, IP-telephony, video-conferencing) require or at least benefit from Quality of Service (QoS) [1,2] or other form of prioritisation that guarantees an Internet connection. Increasing bandwidth is not always sufficient to accommodate these increased demands. QoS mechanisms provide expected and predefined service guarantees by better managing the available bandwidth.

The Differentiated Services (DiffServ) architecture [3,4,5,6] is nowadays the preferred architecture, which can address quality of service issues in IP networks. It provides a coarse and simple way to categorize and prioritize network traffic (flow) aggregates, leaving complexity at the "edges" and keeping the "core" network simple

enabling its scalability. Edge devices (ED) in this architecture perform packet classification, policing, shaping and marking in order to ensure that individual user's traffic conforms to the specified traffic profiles and aggregate traffic into a small number of prioritized classes. Core routers treat packet aggregates with Per-Hop-Behavior (PHB) [7,8] according to their markings. PHB is the forwarding treatment that a packet receives at a network node. The concept of the Bandwidth Broker (BB) Architecture [9,10] was proposed by Internet2 in order to provide an overall resource management, policy-based admission control and configuration of specific network elements (leaf, core and border routers).

Our proposed architecture is based on the DiffServ and BB [11,12] concept. It is basically a realization of a distributed BB architecture, promising scalability and efficiency. Consequently, an additional layer on "top" of the DiffServ architecture is realized, the Resource Control Layer (RCL) as described in the [13]. The RCL is composed of different distributed entities each one assigned a specific task. The algorithm realized and evaluated in this paper is responsible for the resource management performed by the RCL. The rest of the paper is structured as follows: in the following part an outline of the proposed architecture is presented. In the last section, the implemented algorithm is shortly described and evaluated.

MOTIVATION & PROPOSED ARCHITECTURE

The architecture proposed aims at an efficient management and distribution of resources between the different nodes of a DiffServ architecture. This is basically realized by the proposed algorithm implemented in this layer, which achieves a good utilization of network resources. The architecture is fully analyzed in [13], and here its main functionality is described. It is composed of three logical entities. To start with, the Resource Control Agent (RCA) is the highest control entity in an administrative domain and is responsible for configuring the appropriate network entities and managing the network resources. Moreover, it has the overall view of the policies enforced in a domain and decides for the management of bilateral Service Level Agreements between adjacent administrative domains. Second, the Admission Control Agent (ACA) performs admission control based on the traffic profile between the user and the network. In this way, it controls the access of the user to the network and performs authorization and usage metering (accounting) functions. Last, the End-User Application Toolkit (EAT) provides a graphical interface to end-user applications and enables them to signal their requirements to the QoS infrastructure. The above logical entities can be distinguished in Fig. 1.

In order the RCA entity to manage more efficiently the resources distributed among the networks elements, a hierarchical architecture inside the RCA is proposed. Therefore, instead of having a centralized resource management entity, a distributed one is proposed, separating the network to sub-networks. Each sub-area has its own initial resources, which are assigned according to traffic loads forecasts and/or results retrieved by a measurement-based platform. The structure of the RCA is depicted in Fig. 2.

The resources assigned to the administrative domain (root) are distributed among the sub-areas, each one represented by a Resource Pool (RP). Moreover, each sub-area can also be further divided into sub-areas, forming the above hierarchical structure. Another reason for the creation of RPs is the correct management of bottleneck links and the efficient sharing of its bandwidth between the RPs of the lower level. The Resource Pool Leafs (RPLs) correspond to the resources assigned to each ACA. Each ACA is based on those resources to perform admission control. The assignment of the resources is a top-down procedure, from the root of the tree down to the RPLs. On the left hand of the Fig. 2 is given an example of RPs creation based on the network of Fig. 1 and on the right hand a more complicated hierarchical structure.

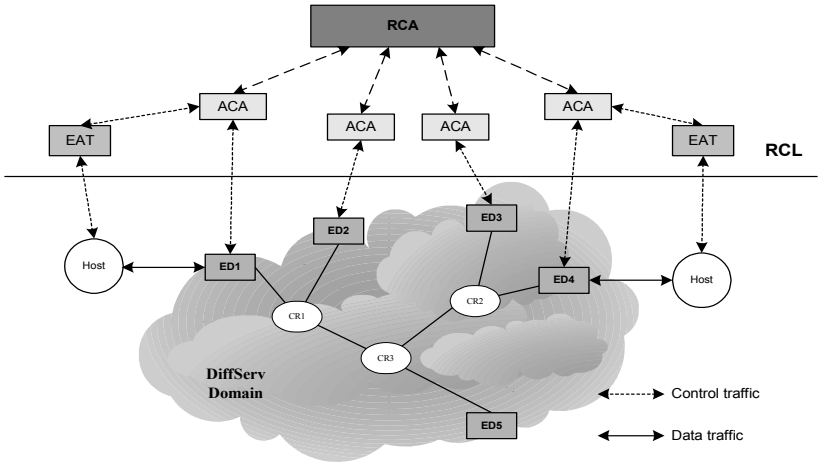

Fig. 1.: RCL infrastructure

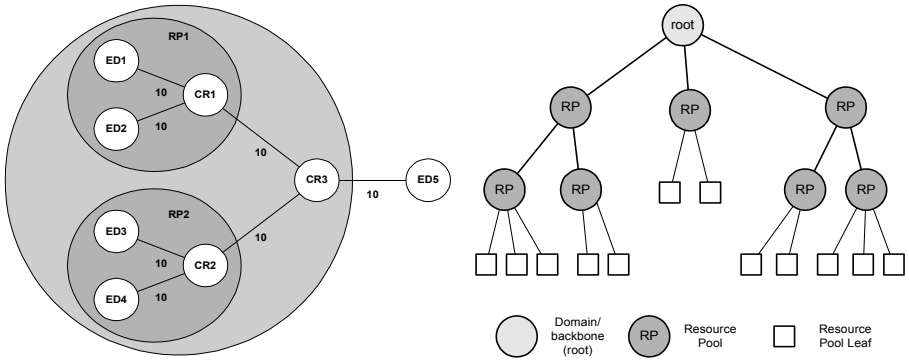

Fig. 2.: Hierarchical structure of RCA

The initial assigned resources may not correspond to actual traffic load, therefore, the RPLs/RPs are capable of adjusting and adapting those initial resource assignments to real traffic conditions, which are difficult to be forecasted and may change during time.

THE ALGORITHM

The main target of the algorithm is to efficiently handle the re-distribution of resources. This is invoked when an RPL does not have enough resources to accommodate a new user request. Each RP and RPL is basically described by the following set of parameters:

R_{max} *: upper limit of resources that can be assigned to an RP/RPL.*
R_{tot} *: current resource assignment to an RP/RPL*
R_{res} *: current reserved resources of an RP/RPL*
R_{free} *: currently free resources of an RP/RPL*
R_{add} *: maximum resources that can be additionally assigned to an RP/RPL*

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

$$R_{max} \geq R_{tot} \geq 0 \quad (1)$$
$$R_{free} = R_{tot} - R_{res} \quad (2)$$
$$R_{add} = R_{max} - R_{tot} \quad (3)$$
$$R^{f}_{res} = \Sigma R^{c}_{tot} \quad (4)$$
$$R^{f}_{max} \geq R^{c}_{max} \quad (5)$$
$$\Sigma R^{c}_{max} \geq R^{f}_{max} \quad (6)$$

The network administrator is responsible for defining the initial resources to be distributed to the nodes of the tree. After this top-down start-up procedure, initial resources are assigned to all nodes of the tree. Sequentially a user can make 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 RPL for admission control.

According to the algorithm realized, an RPL will make a request for additional resources to its father when its current free resources are not adequate to serve a new request. The child makes a request and 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 (R_{add}) 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 RP (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.

A number of additional parameters must be defined for the realization of the algorithm:

R_{req} *: minimum resources requested from an RP/RPL*
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 RP increases the resources of its child by* $A_{max} \times R_{req}$

A_{med} : *number of med resources shifts; father RP increases the resources of its child by* $A_{med} \times R_{req}$

A_{min} : *number of min resource shifts; father RP increases the resources of its child by* $A_{min} \times R_{req}$ $(A_{max} > A_{med} > A_{min} \geq 1)$

ρ_L : *a low limit for the free resources of the RP,* $\rho_L < 1$

ρ_H : *a high limit for the free resources of the RP,* $\rho_H < 1$ $(\rho_H > \rho_L)$

The ρ_L and the ρ_H determine two limits for the free resources of an RP. Actually a low and a high watermark are defined corresponding to $\rho_L \times R_{tot}$ and $\rho_H \times R_{tot}$.

As long as the RPL has enough resources to accept a reservation request, there is no need of redistribution of the resources. In case an RPL does not have efficient resources to accommodate an R_{req} it asks more resources from its father RP, 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 RPL after a resource reservation request are:

1. ***if*** $R^{RPL}_{res} + R_{req} > R^{RPL}_{max}$ ***then reject*** *the request;*

2. ***if*** $R^{RPL}_{res} + R_{req} \leq R^{RPL}_{tot}$ ***then*** *admit request* $R^{RPL}_{res} = R^{RPL}_{res} + R_{req}$
end then (2)

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

In case a father RP can not assign to its child not even the minimum amount of resources requested, it requests in the same way resources from its corresponding father. The father RP uses the following algorithm in order to calculate the resources to give back to its child. The father RP basically compares its low and high watermark of free resources with a multiple of the resources requested. Depending on the result of the comparison, it gives back an appropriate multiple ($A_{max}/A_{min}/A_{med}$) of the resources requested.

1. ***if*** $A_{max} \times R_{ask} < \rho_L \times R_{free}$ ***then*** $R_{recv} = min(A_{max} \times R_{ask}, R^{c}_{add})$, $R_{res} = R_{res} + R_{recv}$
return R_{recv} ***end then (1)***

2. ***else if*** $A_{med} \times R_{ask} < \rho_H \times R_{free}$ ***then*** $R_{recv} = min(A_{med} \times R_{ask}, R^{c}_{add})$, $R_{res} = R_{res} + R_{recv}$
return R_{recv} ***end then (2)***

3. ***else if*** $A_{min} \times R_{ask} < R_{free}$ ***then*** $R_{recv} = min(A_{min} \times R_{ask}, R^{c}_{add})$, $R_{res} = R_{res} + R_{recv}$
return R_{recv} ***end then (3)***

4. ***else*** *ask resources from its father*
$R'_{ask} = (R_{res} + R_{ask}) - R_{tot}$
$R'_{recv} = request(R'_{ask}, R_{add})$
if *request accepted by father* ***then*** $R_{tot} = R_{tot} + R'_{recv}$, *goto step(1)* ***end then***
else *reject the request;* ***end then (4);***

When the ACA makes a release request to the RPL, the latter de-allocates the corresponding resources and checks whether or not it can give back any free resources to its father. In order to take such decision an additional set of variables are defined:

l : *a low limit of the* R_{tot} , $l<1$

R_{rel} : *requested resources to be released*

R'_{rel} : *resources to be given back to the upper level*

a : *it determines indirectly the actual amount of resources to be returned,* $a<1$

The low watermark, *l x Rtot*, is used to check the current status of reserved resources of an RP/RPL. In case the reserved resources before the release are above the low watermark and the resources after the release are below this watermark, then an amount of free resources should be returned to the upper level. The purpose of this double check of resources is to control that an RP/RPL is not actually in an initial state, where resource reservations have just began. In that case its reserved resources may not have yet exceeded the low watermark so that resources should not be returned to the upper level. The amount of resources to be given back should be calculated considering the trade-off between giving as much as possible and keeping resources for future use. This calculation is actually based on the desired level of reserved resources between the total resources and the low watermark. The value of a determines this level.

The algorithm for deciding and calculating the resources to be returned is:

1. *After the release:* $R'_{res} = R_{res} - R_{rel}$
2. ***if*** $(R'_{res} < l \times R_{tot})$ ***and*** $(R_{res} > R_L)$
   then *have to give back resources to the upper level so that reserved resources to be between the* R'_{tot} *and* $l \times R'_{tot}$*:*
   $R'_{res} = a\,(R'_{tot} + l \times R'_{tot})$, *where* $R'_{tot} = R_{tot} - R'_{rel}$
   From above: $R'_{rel} = R_{tot} - R_{res} / (a\,(1 + l)$ ***else*** *do not give back resources*
   end then (2)

SIMULATION

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, as depicted in Fig. 3. The actual tree structure does not play a crucial role for the study of the proposed algorithm.

A simulation experiment consists of a random process of reservation request arrivals. Each request arriving to an RPL may be admitted or rejected according to the specifics of the algorithm in question. The inter-arrival time of reservation requests follows an exponential model, while the size of the resources requested have a standard capacity of 128kbps. Each leaf node has a weight, which determines the amount of initial resources assigned to it. Those initial resources in a real network could have been based on some load forecasts. The offered load to the leaf nodes differs from the one forecasted in order to prove the adaptability of the algorithm. While the resources are distributed to nodes 1,2,3 with weights 0.5, 0.3, 0.2, the actual

offered load is correspondingly 0.5, 0.4, 0.1 for the half time of simulation time and 0.5, 0.2, 0.3 for the rest time.

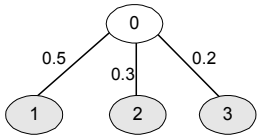

Fig. 3: Simulation topology

In order to verify the performance achieved by the proposed algorithm, it is actually compared to a static configuration, where the concept of resource pools is not used. An amount of resources is assigned to each ACA, which do not change during simulation. Moreover the behavior of the proposed algorithm has been examined under different set of values of parameters. TABLE I summarizes those parameters and assigns to them a possible value.

TABLE I

MAIN VARIABLES OF THE ALGORITHM

Variable	Value
A_{max}	5(3-8)
A_{med}	3(2-4)
A_{min}	1(1-3)
w_L	0.2(0.2-0.5)
w_H	0.6(0.5-0.8)
l	0.5(0.5-0.7)
A	0.5

Primarily we have examined the variation of *Rtot* and *Rres* in time for all the RPs/RPLs of the tree, changing the values of the parameters in TABLE I. In general the algorithm offers an exceptional adaptability as indicated in Fig. 4 for an RPL and *Amax* is set to the value of 7. The adaptability of *Rtot* to the reserved resources, *Rres*, depends mainly on the values of *Amax* and *l*. The greater the value of *Amax* 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.

In sequence the number of interactions among all nodes of the tree was examined for different values of *Amax*. As a result of the simulations the greater the value of *Amax* the smaller the number of interactions.

Another crucial characteristic for the performance of the proposed algorithm is the utilization of the network resources. The average utilization has been measured for each leaf varying the value of *Amax* from 3 to 8, as illustrated in Fig. 5. The algorithm really provides a high utilization, which is inversely proportional to the value of *Amax*. The current utilization of resources of each node depends also directly on the value of *l*, since *l* composes an under bound for the utilization.

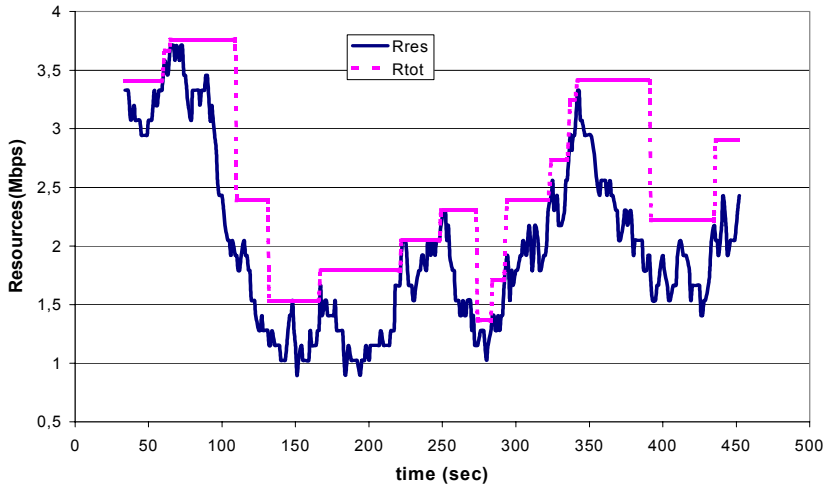

Fig. 4: Status of Resources of a RPL

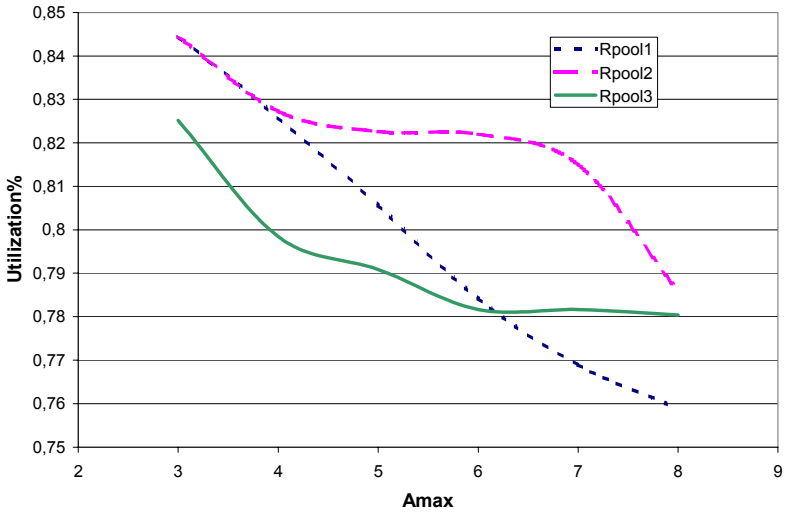

Fig. 5: Utilization in relation to Amax

It has been also examined the response of the algorithm to the modification of values of the other parameters. *Amin*, *Amed*, w_H and w_L also influence the utilization and the number of interactions in the same way as *Amax*, but they have a smaller impact than *Amax*. In addition the behavior of the parameter *a* is identical to that of *l*, since they both determine the state that release of resources should take place.

Finally the number of rejected resource requests has been measured for the proposed as well as the static algorithm, as depicted in Fig. 6. The nodes 1 and 3

under the proposed algorithm invoke no rejections while node 2 (RPL2) generates a small number of rejections. The nodes under the static algorithm generate a number of rejections, which are proportional to the offered load. It is really obvious how the proposed algorithm outperforms the static version, offering a really smaller number of rejections, since it achieves a dynamic resource distribution between the leaves of the tree.

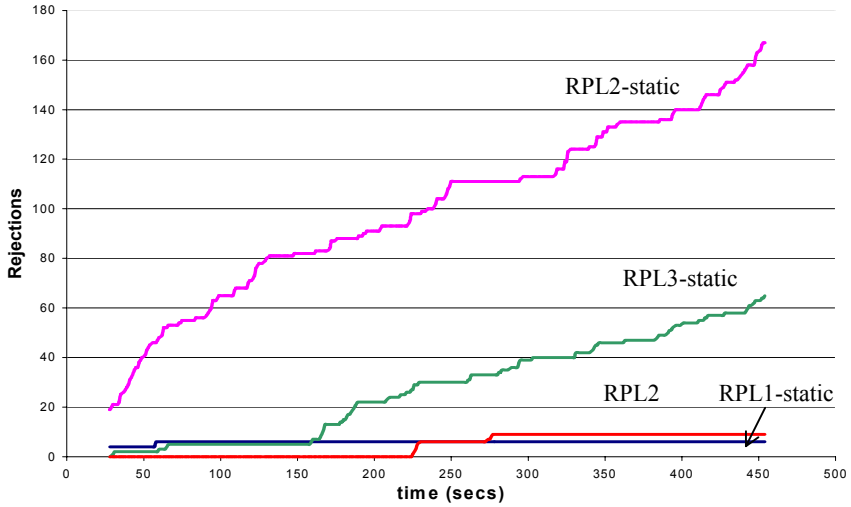

Fig. 6: Number of rejected resource requests

Summarizing, there is trade-off between the utilization 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 *Amax* is required. Consequently, a smaller utilization of network resources is achieved. It depends also on the network administrator to tune appropriately the value of *Amax* and the other parameters in order to achieve the desired performance. In addition it has been verified the significant improvement in bandwidth assurance and resources utilization of the proposed algorithm compared to a static version, which keep though the complexity at a really low level.

CONCLUSIONS & FUTURE WORK

The proposed realized algorithm uses some techniques in order to adapt efficiently and dynamically the resources of an RP/RPL to real traffic loads. The simulation results prove how this algorithm outperforms a static configuration, without a significant complexity burden.

A management platform is under study in order to provide a graphical interface for the monitoring and configuration of the RPs. In addition new versions of the proposed algorithm are planned for the future in order to examine more the role of the different parameters as well as to tune their value more properly.

ACKNOWLEDGEMENTS

This paper is partly funded by the European research project AQUILA, Information Societies Technology (IST) programme, IST-1999-10077. The authors are currently engaged in the definition, implementation and evaluation of the concept presented in this paper.

Reference

[1] Ben Teitelbaum et al, “Qbone Architecture (v1.0)”, Internet2 QoS Working Group Draft, August 1999, Work-in-Progress.

[2] Ben Teitelbaum,Ted Hanss,“QoS Requirements for Internet2”,April 22, 1998.

[3] Black, D.; Blake, S.; Carlson, M.; Davies, E.; Wang, Z.; Weiss, W., “An Architecture for Differentiated Services”, RFC 2475, 1998.

[4] Y. Bernet, J. Binder, S. Blake, M. Carlson, S. Keshav, E. Davies, B. Ohlman, D. Verma, Z. Wang, W. Weiss, "A Framework for Differentiated Services", Internet Draft, October 1998.

[5] Ben Teitelbaum, “Qbone Architecture (v1.0)”, Internet2 QoS Working Group Draft,August,1999, http://www.internet2.edu/qos/wg/papers/qbArch/1.0/draft-i2-qbone-arch-1.0.html.

[6] D.Grossman,"New Terminology for Diffserv",Internet-Draft,November 1999.

[7] V. Jacobson, K. Nichols, K. Poduri, "An Expedited Fowarding PHB", IETF Proposed Standard RFC 2598, June 1999.

[8] Heinanen, J., Baker, F., Weiss, W. and J. Wroclawski, "Assured Forwarding PHB Group", RFC 2597, June 1999.

[9] K. Nichols, V. Jacobson, L. Zhang, “A Two-bit Differentiated Services Architecture for the Internet”, RFC 2638, July 1999.

[10] R. Neilson, J. Wheeler, F. Reichmeyer, S. Hares, “A Discussion of Bandwidth Broker Requirements for Internet2 Qbone Deployment”, Internet2 Qbone BB advisory Council, August 1999.

[11] J. Ogawa, Y.Nomura, “A Simple Resource Management Architecture for Differentiated Services”, Internet Society, Japan, July 2000.

[12] British Columbia Institute of technology, "CA*net II Differentiated Services-bandwidth Broker High-Level Design", November 1998.

[13] G. Politis, P. Sampatakos, I.S. Venieris, "Design of a multi-layer bandwidth broker architecture", Lecture Notes in Computer Science; Vol 1938, Springer Verlag, Oct 2000.