Using the ns-2 simulation environment to implement and evaluate Bandwidth Broker models

Christos Bouras, Ioannis Pappas, Dimitris Primpas and Kostas Stamos
Research Academic Computer Technology Institute, N. Kazantzaki Str., University of Patras, 26500 Rion, Greece &
Department of Computer Engineering and Informatics, University of Patras, 26500 Rion, Patras, Greece
Tel: +30 2610 {960375, 996954, 960316, 960316}
Fax: +30 2610 960358
email: bouras@cti.gr, pappasj@ceid.upatras.gr, primpas@cti.gr, stamos@cti.gr

***Abstract*-- In this paper we present and compare four different models of implementing a Bandwidth Broker in a DiffServ network. We describe the relevant implementation aspects in the well-known ns-2 simulator and present a number of experiments that were conducted in order to verify our implementation, analyze the performance characteristics of the various alternatives and evaluate the usefulness of the ns-2 simulation environment for similar purposes. We present our experiments that aim at investigating a number of issues related to the operation of Bandwidth Broker modules, and in particular how the acceptance rate, the network overhead and the response time for each Bandwidth Broker model are affected by the network topology, the available buffers for incoming requests and the advance time from the moment a request is submitted to the moment the reservation should take place.**

Keywords—*bandwidth broker; QoS; dimensioning; ns-2 simulator*

I. INTRODUCTION

In order to improve over the traditional best-effort service that is typically offered today for all kinds of Internet traffic, a number of Quality of Service (QoS) architectures have been proposed. The most widely used architecture is DiffServ [1], which minimizes the number of actions to be performed on every packet at each node and builds a configuration that, unlike the alternative IntServ architecture, does not use a signaling protocol. Individual DiffServ mechanisms are applied on traffic aggregates rather than individual flows. The operation of the DiffServ architecture is based on several mechanisms. The first mechanism is the classifier that tries to classify the whole traffic into aggregates of flows (traffic classes), mainly using the DSCP field (Differentiated Service CodePoint [15]). This field exists in both the IPv4 and IPv6 packet headers, as part of the Type of Service (ToS) field and as part of the Traffic Class field, respectively. The operation of services based on DiffServ architecture uses also several additional mechanisms (packet marking, metering and shaping) that act on every aggregate of flows. In addition, in order to provide QoS guarantees it is necessary to properly configure the queue management and the time routing/scheduling mechanism. The most common queue management approaches use the Priority Queue, Weighted Fair Queue or Modified Deficit Round Robin mechanisms.

The Bandwidth Broker [5] is an entity that manages the resources within a specific DiffServ domain by controlling the network load and by accepting or rejecting bandwidth requests. Every user (service operator) who is willing to use an amount of the network resources, between its node and a destination, sends a request to the Bandwidth Broker. For requests that span multiple domains (inter-domain requests), the Bandwidth Broker will have to communicate with Bandwidth Brokers in the adjacent domains that are traversed by the requested flow. Bandwidth Brokers only need to establish relationships of limited trust with their peers in adjacent domains, unlike schemes that require the setting of flow specifications in routers throughout an end-to-end path. Therefore, the Bandwidth Broker architecture makes it possible to keep state on an administrative domain basis, rather than at every router and the DiffServ architecture makes it possible to confine per flow state to just the leaf routers. Bandwidth Brokers are an intensely studied field and a number of architectures have been proposed for the various aspects of its operation ([9], [10], [11], [12], [13], [14]).

In order to study the Bandwidth Broker operation and the possible choices in terms of deployment and network management, we have used the ns-2 simulation environment [6] and implemented four variations of Bandwidth Broker architectural models. These models have been used in order to extensively test a number of aspects of their operation and isolate the architectural choices that influence in a specific way the efficiency of each model. The rest of the paper is structured as follows: Section 2 describes our ns-2 implementation and gives an overview of the Bandwidth Broker models that we have tested, while section 3 presents the setup of the experiments. Section 4 deals with the effect and importance of buffer usage for storage of incoming requests and section 5 presents our results throughout a number of different network topologies. Finally, section 6 presents our conclusions and the future work that we intend to do in this area.

II. BANDWIDTH BROKER IMPLEMENTATION IN NS-2

The four architectural models that have been developed, implemented and tested in the ns-2 simulator environment are the following:

Serial Distributed Bandwidth Broker model (SDBB model)

Parallel Distributed Bandwidth Broker model (PDBB model)

Centralized Bandwidth Broker model (CBB model)

Centralized Fault-tolerant Bandwidth Broker model (CFBB model)

In every implementation, two kinds of agents exist, the Edge Bandwidth Broker (BBedgeAgent) and the Base Bandwidth Broker (BBbaseAgent). An agent in ns-2 is an endpoint where packets are consumed and constructed using a specific protocol. A BBedge agent is a simple module located at each node (router) of the network that represents a client (a user or an application) and its function is to send requests for traffic with specific profile for a specific time period. Such a request is received by the BBbase agent, which represents a server, and added as a request with specific parameters (sender, end node, bandwidth, time limit and status, which at this stage is set to pending) at a local database. When the processing of the request finishes, the status request changes according to the result of the processing, either to "satisfied" or to "rejected". The answer is then sent back to the BBedge agent.

Another common element of the BBbase agents for all models is the request buffer. The BBbase agents are processing one request at a time so if two requests arrive closely to each other, the first one is going to be served and the other one is going to be added at a buffer that exists at the BBbase agents. If a request arrives and the buffer is empty, the request is immediately processed. Otherwise, if the buffer contains previous requests but is not full yet, the request is added at the end of the buffer in order to be processed in the future. Finally, if the buffer is full, the request is rejected. The length of the buffer is configurable and it is a point of investigation for the efficient operation of a bandwidth broker.

Although the structure of the BBedge agents is the same for all four models that have been implemented, each one operates in its own fashion when various packets are received during the simulation. The differences between the four architectures have mainly to do with the structure and the implemented behavior protocols. The different models use different methods to process the requests (admission control) and to store the necessary information. The request processing methods are described in detail in the next section, while the remainder of this section deals with the storage of information.

The SDBB and PDBB models store the information about the state of each link (reserved bandwidth for existing requests, remaining bandwidth) at the corresponding BBedge agents. On the contrary, the CBB model stores this information at the BBbase agent and the CFBB model stores the information both at the BBedge agents and at the BBbase agent. The format of this information is the following: Consider for example node 1 that is connected with node 2 and node 2 is connected with node 1. The relevant information for node 1 is that "The available bandwidth from node 1 to node 2 is b for the time period t2-t1", where t2, t1 are points in time, and b is the bandwidth metric that arises from dimensioning of the network and the service. The relevant information for node 2 is that "The available bandwidth from node 2 to node 1 is b' for the time period t'2-t'1", where t'2, t'1 are points in time, and b' is the bandwidth metric that also arises from dimensioning of the network and the service. In our simulation model we make the assumption that b' equals b, t'2 equals t2 and t'1 equals t1 between two nodes, in other words that all requests and reservations are bi-directional. For the CBB and CFBB models, the BBbase agent's local database has, per each node it manages, the complete information of available bandwidth for every link to a neighbouring node for all time moments.

A. The supported QoS service

The Bandwidth Broker provides a QoS service with the characteristics of bandwidth guarantee as well as minimum delay and jitter. This service is the IP Premium and is currently supported by many network providers. The main characteristic of this service is that it follows the classic DiffServ architecture. It classifies the packets using the DSCP values for admitted and downgraded packets. The policing is performed at the edge of the network and high priority queuing is applied in the core and access routers at the outgoing interfaces.

The original ns-2.26 functionality supports packet classification at the edge routers using the source-destination pair of the IP header. We have already enhanced the simulator so that the classification is done using the DSCP field of the IP header [7]. This enables packets that have the same source and destination nodes but belong to different applications to belong to different classes as well, and packets with different source and destination nodes to belong to the same class.

The QoS service has the responsibility of packet classification and policing. If the BBedge agent receives a positive answer about a request it has submitted, it configures through tcl code all the edges that exist on the request path. After the configuration process has been completed, the BBedge agent can start using the requested and allocated network resources.

The QoS implementation starts with the insertion of the DSCP value into the packet headers for packets that use the requested service. When these packets are inserted into the network with the proper DSCP value, strict token bucket policy is applied to them, when they are in the first BBedge agent. This action guarantees that the transmitted rate matches

the requested (admitted) rate. Next, the queue management mechanism is properly configured. The used queue management mechanism is a high priority queue on every node, which is used for all the admitted traffic classes.

B. Description of the implemented models

In each one of the implemented models, we use a different communication protocol in order to complete the processing of requests. Our purpose is to simulate, as accurately as possible, a network that is managed by a Bandwidth Broker. Communication between a BBbase agent and a BBedge agent is achieved with the use of messages that are always sent either from a BBedge to a BBbase or from a BBbase to a BBedge agent. Two BBedge agents never communicate by sending messages to each other, since the BBbase agent always intervenes in the communication.

For the SDBB model the processing takes place as follows. In this model, all the information about the status of links regarding available bandwidth is stored locally at the BBedge agents, as has already been mentioned. At the beginning, the BBedge sends a bandwidth request to the BBbase agent. When the BBbase receives the request, it stores the address of the sender. Then, using an ns-2 simulator's tcl command ("lookup"), it finds the neighbor node of the request sender. In particular, by inserting the start and the end node of the path, ns-2 simulator uses the OSFP protocol to find the shortest path between these two nodes and then returns to the user the first node of this path that is located next to the request sender. So, at first, the BBbase sends a packet to the BBedge agent that initiated the bandwidth request, querying whether there is available bandwidth from itself till its next neighbor node. If the BBbase receives a positive answer from the first BBedge agent of the path, then the BBbase asks the neighbor of the first node for bandwidth and so on, until the BBbase receives a negative answer, in which case it stops the processing of the request, sends negative answer to the request sender and goes on with the next request, or until the BBbase gets positive answers from all the nodes which are located on the request path, in which case it sends positive answer to the request sender and a packet to each node on the request path, in order to update their information about bandwidth availability. After that, the BBbase moves on to process the next request (if any). A positive answer means that the BBbase agent allows the request sender to use the requested bandwidth by guaranteeing the appropriate resource reservation and a negative answer means that the request was rejected because of lack of bandwidth.

The PDBB model, similarly to the SDBB model, stores information at the BBedge agents too but it has a different way of retrieving this information in the course of processing a request. When the request arrives, the BBbase agent sends a packet to each one of the nodes that are located on the request path querying them for bandwidth at a particular time period. If it receives even a single negative answer, it does not wait for any further response by BBedge agents, but rather it stops the request process, sends a negative answer to the request sender and moves on to another request. If the BBbase agent keeps getting positive answers from the nodes, it waits until all the nodes respond to its query. If all the nodes have answered positively, then the rest of the request process is identical as for the SDBB model. The main difference from SDBB is therefore that now the BBbase agent doesn't sequentially query the nodes, but concurrently (sort of flooding the query messages) and waits until all the BBedge agents respond. The PDBB model aims at reducing response times for requests over the SDBB model by parallelizing the queries over the network.

The CBB model is a centralized model compared to the distributed nature of previous models. The BBbase agent stores all information about the status and availability of the links at a local database, which it consults in order to process a request. The response time is therefore reduced by eliminating much of the network communication, but the CBB model also has reduced resiliency because of its dependence on a single node.

The CFBB model combines the advantages of the CBB model and adds some information redundancy for the purpose of increased resilience to node failures. Data regarding the link status and availability is stored both at BBedge and BBbase agents. An incoming request is processed exactly as in the CBB model, but upon a positive answer, the BBbase agent not only updates its local data base, but it also sends a packet to each node on the request path to update their relevant information. This information is not retrieved during the request processing, but it can be used in the case of a failure at the node hosting the BBbase agent. Another node can then run a new BBbase agent that is brought up-to-date by all the BBedge agents (each sending a single message to the new BBbase agent with the relevant information) about the availability of bandwidth and the current reservations. The trade-off for the CFBB model is the generation of some additional network overhead over the CBB model.

III. EXPERIMENTAL SETUP

The Bandwidth Broker models' performance was compared in a number of different network topologies using several criteria and metrics. An important metric is the ratio of positive answers (the requests that were accepted and the appropriate resources were reserved) relative to the total number of submitted requests for each experiment. We also studied the response time (the time from the moment a request is submitted to the BBbase agent until the moment the BBbase agent responds either by accepting or rejecting it) and the network overhead caused by the control messages exchanged between the BBbase and BBedge agents.

Requests in our experiments were randomly generated by an ns-2 simulator's tcl script that was using the rqst variable to determine the number of requests to be generated. The parameters for each request were randomly produced within suitable boundaries (regarding the total duration of each simulation, the total available bandwidth, the minimum and

the maximum reservation requests) for each situation that we wanted to simulate. Specifically, we produced random requests with random request send time, start time, stop time, bandwidth and neighbor id. The requests had specific time limits regarding their duration (stop time minus start time), much smaller than the total duration of each experiment, in order to offset any initialization effects.

Randomness was obtained by using the ns-2 RNG class. This class contains an implementation of the combined multiple recursive generator MRG32k3a [8]. The MRG32k3a generator provides 1.8x1019 independent streams of random numbers, each of which consists of 2.3x1015 substreams. Each substream has a period (i.e., the number of random numbers before overlap) of 7.6x1022. The period of the entire generator is 3.1x1057, thus more than adequate for generating randomness for our purposes. The random generator was independently generating numbers that were then assigned to each of the attributes for a new request. If the random combination of attributes was valid (e.g. the stop time was not earlier than the start time), the request was generated by the node and sent to the Bandwidth Broker for processing.

IV. STUDYING THE EFFECT OF BUFFER SIZE

In our first experiment, we studied how the buffer requirements of the BBbase agent affect the overall performance and in particular how they are related to the percentage of accepted requests. We only tested the SDBB and PDBB models since they are the only ones that have to use the network in order to complete the processing of a request, while the centralized models (CBB, CFBB) perform the request processing internally and as a result much faster and therefore have much lower need for buffering. The network topology used for this experiment was a serial one with the BBbase agent located at the middle, as shown in Figure 1.

The request buffer for the SDBB and the PDBB models stores requests that cannot be immediately processed. If, as soon as the request is processed, its start time has passed, the request is rejected.

The horizontal axis in Figure 2 displays the size of the buffer in logarithmic (ln) scale and the vertical axis measures the percentage of accepted requests. When there is no buffer (zero size), there is a large performance hit, especially for the SDBB model, but using a buffer size of ten or greater, the negative effect is greatly reduced. For the PDBB model this means that with the addition of a very small buffer to the agents, no requests are lost due to unavailability. This conclusion can also be reinforced by the fact that at similar experiments, the CBB/CFBB models also displayed similar performance (20% accepted requests). For the SDBB model, although the buffer improves the situation, the performance never quite reaches the acceptance rate of the rest of the models, because of the linear and therefore slow nature of the SDBB model. Even when the buffer practically becomes infinite (i.e. no requests are dropped because of lack of buffer space), the slow operation of SDBB forces many requests to be dropped simply because their start time has expired by the time they get to be processed.

It has to be noted that the links in the network were simulated with a latency of 1ms, which is a typical to relatively low value for non-local networks. Larger latency values would obviously have further detrimental effect to the performance of SDBB.

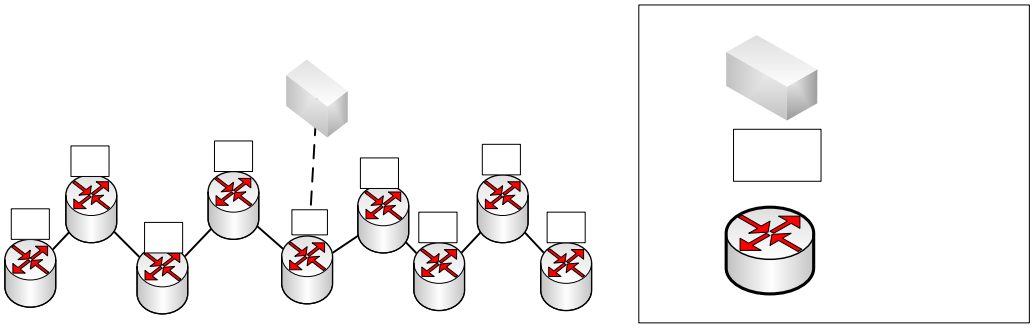

Figure 1. Serial topology

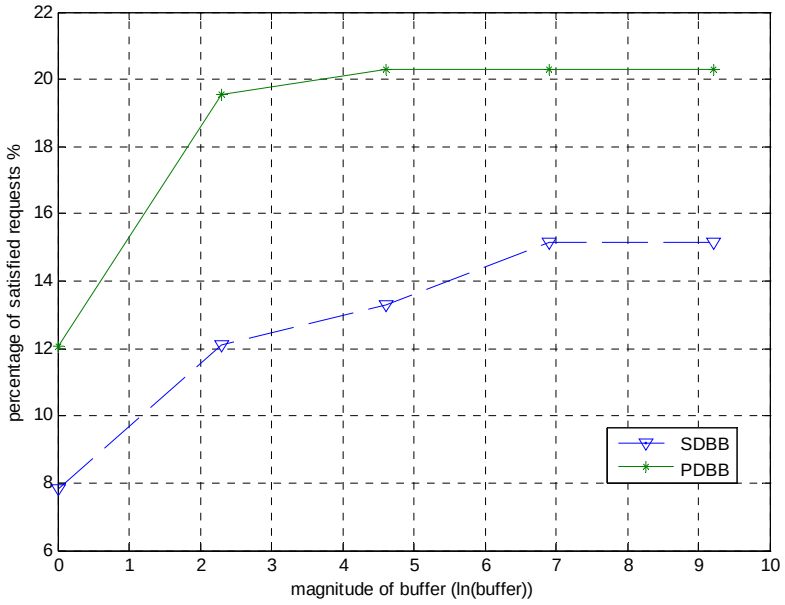

Figure 2. Acceptance rate vs. the size of request buffer (link latency 1ms)

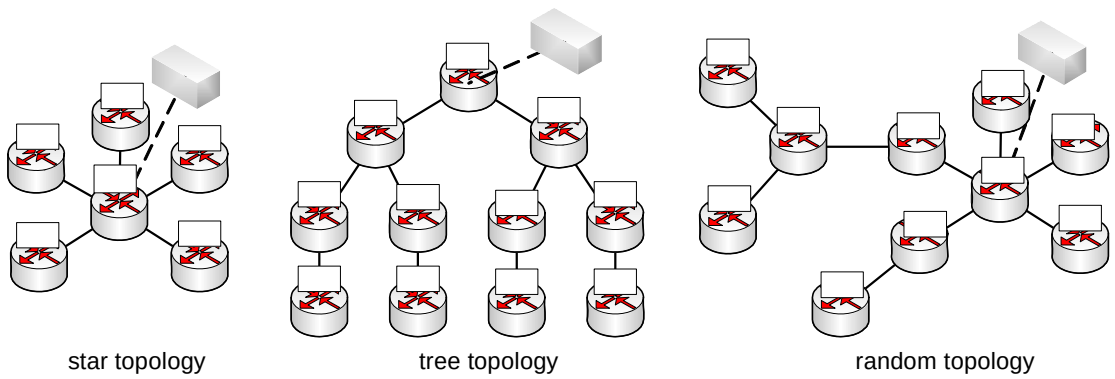

Figure 3. Additional topologies used for experiments

V. EVALUATION OF ARCHITECTURES USING VARIOUS TOPOLOGIES

For our main set of experiments we used a number of different topologies in addition to the basic topology of Figure 1.The additional topologies are displayed in Figure 3, and they were designed so that many possible characteristics of the network topology can be compared and taken into account. For example, while the star topology minimizes the distance between the BBbase and the BBedge agents, the tree and serial topologies maximize it and magnify its effect on the Bandwidth Broker performance. The tree and serial (which is typically also a tree) topologies' main difference is that the serial topology has more potential bottlenecks than any other topology, since most requests are probably going to request part of the resources of the middle links. Finally, the random topology combines characteristics of both the star and the serial and tree topologies in order to also have a more balanced set of results. For all experiments, unless otherwise noted, the latency of all links was set at 1ms.

C. *Network overhead*

In Table I we have summarized the network overhead caused by each Bandwidth Broker model for each topology, measured by the average number of packets exchanged per each request.

Because of their distributed nature, there is much more network overhead for the SDBB, PDBB compared to the CBB and CFBB models, especially for the serial and tree

topologies. This difference is significantly reduced for the star topology, because the star topology fits better to the implementation algorithm of the SDBB, PDBB models. In all cases the CFBB model represents a "middle of the road" alternative between the SDBB/PDBB and CBB models.

Also, those experiments show that the overall network overhead is affected by the topology and the location of the BBbase agent combined with the distribution of the QoS requests across the network nodes. It is a well known problem that is addressed by studying the topology, the distribution of the QoS usage and therefore applying periodic optimal selection of host node for the BBbase agent.

TABLE I. NETWORK OVERHEAD (AVERAGE NUMBER OF PACKETS PER REQUEST)

Model \ Topology	Serial	Star	Tree	Random
SDBB	7.65	4.76	7.39	5.99
PDBB	9.68	4.92	9.92	6.61
CBB	2.01	1.70	1.94	1.90
CFBB	3.66	2.44	3.63	3.01

D. Acceptance rate

Table II shows the acceptance rate per model for each topology. All models benefit from the star topology and produce better results. The reason is that the links are on the average less loaded at the star topology than the rest topologies, where bottleneck links appear.

TABLE II. ACCEPTANCE RATE (RATIO OF ACCEPTED TO SUBMITTED REQUESTS)

Model \ Topology	Serial	Star	Tree	Random
SDBB	0.1514	0.2216	0.1699	0.2012
PDBB	0.2029	0.2227	0.2171	0.2130
CBB	0.2035	0.2259	0.2088	0.2106
CFBB	0.2060	0.2323	0.2083	0.2118

Generally the differences between the models are rather small, except for the SDBB model which, especially for the topologies with larger average distances displays significantly worse behavior than the rest. Its very large delays in responding to requests cause a lot of them to have expired by the time they are processed. This is also made clear by the average response times which are provided in Table IV. During our experimentation we decided to investigate whether forcing the users to submit requests with a minimum of advance time (a minimum time period from the time a request is submitted to the time the requested resources should be reserved) would help SDBB overcome the problem of reduced acceptance rate. However, because of its very large response times compared to the rest of the models, the arrival rate of requests at the SDBB buffer is faster than the processing rate (given a steady rate of incoming requests, as in our experiments). Therefore, the SDBB model will always eventually fail to process some requests on time (before the moment their reservation should start).

Because of the lack of potential bottleneck links, the star topology allows the largest percent of requests to be accepted. Since in general the topology is typically going to be fixed and the only realistic choice is going to be between the Bandwidth Broker models, Table II suggests that for a random topology the SDBB model is the only one that seems inefficient, while the rest display similar behavior, which hints that this is probably the best behavior one can expect, short of using more sophisticated admission control algorithms [3] or more sophisticated positioning of the BBbase agent [2].

In order to further study the effect of latency, we have repeated the experiments for the SDBB\PDBB models with 10ms latency. Such latency could simulate a Bandwidth Broker operating at a domain that spans remote areas, such as a national network. As can be seen in Table III, such increase of the latency has very detrimental effects to the performance of the distributed models. This leads us to the conclusion that for Bandwidth Brokers operating at wide area domains, the centralized approach is advantageous to the distributed one, at least with the admission control procedures used by the models described in this paper.

TABLE III. ACCEPTANCE RATE FOR LATENCY 10MS

Model \ Topology	Serial	Star	Tree	Random
SDBB	0.0201	0.1263	0.0265	0.0366
PDBB	0.0515	0.2099	0.0714	0.0861

E. Model response time

Table IV presents the average response times for all models in the examined topologies. The response times for the SDBB model are orders of magnitude larger than the rest, which explains it inferior performance in the acceptance rate metric and its unsuitability for all but the most convenient cases (e.g. star topology).

TABLE IV. RESPONSE TIME (AVERAGE TIME PASSED UNTIL THE ANSWER RETURNS TO THE REQUEST SENDER)(MSEC)

Model \ Topology	Serial	Star	Tree	Random
SDBB	1919	4.077	1325	314.3
PDBB	22.19	2.520	9.382	6.041
CBB	5.502	1.597	4.097	2.879
CFBB	5.502	1.597	4.097	2.879

In order to better understand the actual impact of the numbers in Table IV, we have also calculated the standard deviation for the response times, which is presented in Table V. These results show that the SDBB response times not only are much higher than for the rest of the models in average, but that they also fluctuate much more widely. In all cases, the CBB and CFBB models are identical as expected, since their request processing procedures are exactly the same.

TABLE V. STANDARD DEVIATION OF THE RESPONSE TIME (10^{-3})

Model \ Topology	Serial	Star	Tree	Random
SDBB	757	2.180	591.7	224.1
PDBB	19.83	1.485	6.404	4.726
CBB	2.576	0.8128	2.176	1.770
CFBB	2.576	0.8128	2.176	1.770

The close proximity of the nodes to the BBbase agent in the star topology favor the PDBB distributed model which closes the gap to the CBB\CFBB centralized models. But also for the rest of the topologies the response time remains within 2-4 times the time for the CBB\CFBB models, which can be a reasonable trade-off for the distributed advantages of the PDBB model.

VI. CONCLUSIONS – FUTURE WORK

Our main work focused on the implementation and the comparison of four different Bandwidth Broker models using ns-2. The Bandwidth Brokers provide a QoS service to the admitted traffic using well-known DiffServ functionality. The tests that were described above, demonstrate the differences among the Bandwidth Broker agents. Our results clarified the trade-offs that are made between a centralized and a distributed approach, a fault-tolerant and a simple centralized version, and between a serial and a parallelized distributed request processing method. Additionally we saw that with the use of a small buffer at the BBbase agent, we can significantly increase the percentage of positive requests.

In our future work we intend to expand the scope and variety of our implementations, as well as the resulting experimentation. In particular, we intend to enhance the CFBB model with adaptive capabilities that will allow it to be able to relocate the position of the Base Bandwidth Broker agent. This functionality can be useful both in overcoming a failure of the node that hosts the Base Bandwidth Broker agent, and also in relocating the BBbase agent according to its optimal (or approximately optimal) positioning that can be computed as demonstrated in [2]. Furthermore, we intend to analyze more sophisticated admission control algorithms that take into account the network's utilization and achieved acceptance rate [3]. Finally, we intend to study and implement functionality to simulate inter-domain operation, which poses additional challenges, such as the peering model, the pathfinding procedures and their effectiveness, and the SLAs between domains and their dynamic negotiations.

VII. REFERENCES

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

[2] C. Bouras, D. Primpas, "An admission control and deployment optimization algorithm for an implemented distributed Bandwidth Broker in a simulation environment", 4th International Conference on Networking – ICN 2005, Reunion Island, France, 17 - 21 April 2005, pp. 766 - 773

[3] C. Bouras, K. Stamos, "An Adaptive Admission Control Algorithm for Bandwidth Brokers", 3rd IEEE International Symposium on Network Computing and Applications (NCA04), Cambridge, MA, USA, 30 August - 1 September 2004, pp. 243 - 250

[4] N. Duffield, P. Goyal, A. Greenberg, "A flexible model for resource management in virtual private networks", ACM SIGCOMM 1999"

[5] K. Nichols, V. Jackobson, L. Zhang, "RFC 2638: A Two-bit Differentiated Services Architecture for the Internet", July 1999

[6] The Network Simulator - ns-2, http://www.isi.edu/nsnam/ns/ (Accessed February 2006)

[7] http://ru6.cti.gr/diffserv-ns/intro.htm (Accessed February 2006)

[8] Pierre L'Ecuyer. Good parameters and implementations for combined multiple recursive random number generators. Operations Research, 47(1):159–164, 1999.

[9] S. Sohail, S. Jha, "The Survey of Bandwidth Broker", Technical Report UNSW CSE TR 0206, School of Computer Science and Engineering, University of New South Wales, Sydney 2052, Australia, May 2002

[10] "QBone Bandwidth Broker Architecture", QBone Signaling Design Team, http://qbone.internet2.edu/bb/bboutline2.html

[11] T. Braun, G. Stattenberger, "Performance of a Bandwidth Broker for DiffServ Networks", Kommunikation in verteilten Systemen (KiVS03), Leipzig, Germany, March 25-28, 2003

[12] J. Ogawa, A. Terzis, S. Tsui, L. Wang, L. Zhang. "A Prototype Implementation of the Two-Tier Architecture for Differentiated Services", RTAS99 Vancouver, Canada

[13] C. Brandauer, W. Burakowski, M. Dabrowski, B, Koch, H. Tarasiuk, "AC algorithms in Aquila QoS IP network", 2nd Polish-German Teletraffic Symposium PGTS 2002, Gdansk, Poland, September 2002

[14] C. P. W. Kulatunga , J. Kielthy, P. Malone, M. Ófoghlú, "Implementation of a simple Bandwidth Broker for DiffServ networks", Inter-Domain Performance and Simulation IPS 2004, Budapest, Hungary, March 2004

[15] Nichols K. and Carpenter B., "Definition of Differentiated Services Per Domain Behaviors and Rules for their Specification" IETF RFC 3086, April 2001