# Design and implementation of a Bandwidth Broker in a simulation environment

Christos Bouras[1,2], Dimitris Primpas[1,2], Kostas Stamos[1,2] and Nikolaos Stathis[2]

[1]Research Academic Computer Technology Institute, Riga Feraiou 61, GR-26221 Patras, Greece and
[2]Computer Engineering and Informatics Dept., Univ. of Patras, GR-26500 Patras, Greece
Tel:+30-2610-{960375, 960316, 990316}
Fax:+30-2610-{969016, 960358, 960358}
e-mail: {bouras, primpas, stamos}@cti.gr, stathis@ceid.upatras.gr

**Abstract**

A Bandwidth Broker is an entity that has been proposed in order to provide end-to-end QoS. For this purpose it manages the bandwidth (the QoS services) within a network domain and it is also responsible for the communication with Bandwidth Brokers of adjacent domains. In this paper, we present a Bandwidth Broker implementation in the widely used NS-2 simulation environment. We describe the Bandwidth Broker architecture that was our model for the simulation implementation. Furthermore, we illustrate the experiments that we conducted that examined the Bandwidth Broker functionality as well as its performance. All the experiments demonstrated that with this implementation we can achieve end-to-end QoS and that our implementation can be used in order to model the behavior of a Bandwidth Broker in the widely popular NS-2 network simulator.

## 1 Introduction

One of the most important solutions that have been proposed for QoS provision has been the Differentiated Services architecture [1]. The DiffServ framework proposes a scalable service differentiation in the Internet. In order for the DiffServ architecture to be applied, some functional elements must be implemented in network nodes. These elements include a set of per-hop forwarding behaviors, packet classification functions, and traffic conditioning functions including metering, marking, shaping, and policing. There has been a lot of research on developing such functional elements so as to provide better QoS. Currently, many researchers focus on finding ways to provide end-to-end QoS over the Internet. It has quickly become evident that simply using the current DiffServ framework does not solve the problem due to the fact that the Internet consists of numerous network domains acting as autonomous systems, each of which may be differently configured. In order to overcome this problem a bilateral agreement between adjacent domains has to be achieved. One entity that has been proposed to provide end-to-end QoS across a network domain and can obtain synchronization and cooperation with adjacent domains is the Bandwidth Broker [2].

Following this research activity, we are working on the implementation of a Bandwidth Broker on NS-2 [8]. NS-2 is a powerful, open-source simulating tool that provides some basic DiffServ functionality. The DiffServ functionality has already been extended in [3], [9]. This paper describes the implementation issues for the Bandwidth Broker as well as the necessary experiments in order to validate and evaluate our implementation.

The rest of the paper is organized as follows: Section 2 presents an introduction to Bandwidth Brokers. Section 3 gives a small description of the simulation tool, the architecture and the implementation of the Bandwidth Broker in this tool. Section 4 presents the experiments that were conducted and finally section 5 describes our conclusions as well as the future work that we intend to do on this area.

## 2 Bandwidth Brokers

A Bandwidth Broker (BB) [2] is an entity responsible for providing QoS within a network domain. The BB manages the resources within the specific domain by controlling the network and by accepting or rejecting requests for the QoS services that the network provides. Every user (service operator) that is willing to use a QoS service (in our case an amount of the bandwidth), between its node and a destination, sends a request to the BB. The choice of the BB to either accept or reject a request is based on the network load and on the Service Level Agreement (SLA) [7]. The SLA is the service contract between the service provider and every

customer that indicates the service that the costumer is going to receive. The decision to accept or reject a request is made by a module called admission control that takes into account the network condition and the pre-defined SLA. A BB is also responsible for the inter-domain communication with BBs of adjacent domains. This procedure is quite complicated, as it requires direct communication between the 2 adjacent BBs and also a special agreement between the 2 domains that should be taken into consideration by the decision mechanism.

Figure 1 shows a representation of a network containing three domains: AS1, AS2 and AS3. In each domain there is a BB and a number of service users. Every BB communicates with the service users and with its adjacent BBs. A bandwidth broker contains several modules, as Figure 2 illustrates, that are necessary for its transparent and efficient operation.

- An inter-domain interface. It is used for communication with adjacent BBs
- An intra-domain interface. It is used for communication with the service components that are located inside the domain that the BB controls
- A routing table interface. It is used so that the BB knows the network topology and the routing paths
- A user/application interface. The scope of this interface is to allow the user and applications to send requests to the BB.
- A policy manager interface. This interface allows implementation of complex policy management or admission control.
- A network management interface. It is used for coordination of network provisioning and monitoring.

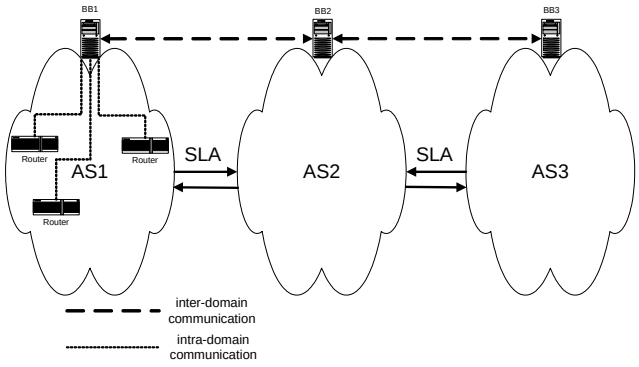


**Figure 1: A network representation containing three domains**

When a user is willing to use an amount of network resources, he sends a Resource Allocation Request (RAR) to the BB of his local domain. The BB receives the RAR, examines whether the requested resources are available and also whether the SLA requirements are fulfilled (this operation is performed by the admission control module) and sends a Request Allocation Answer (RAA) back to the user informing him if the RAR was accepted or not. If the RAA contains a positive answer, the network resources that were requested should be reserved. In order to achieve this, the BB uses the intra-domain interface to configure the routers placed in its domain in order to apply a specified per hop behaviour, according to the QoS service that the network provides.

The resource management in each domain is mainly accomplished via the DiffServ architecture. The DiffServ architecture proposes the provision of service differentiation to the traffic in a scalable manner by suggesting the aggregation of individual application flows with similar quality needs. These aggregations are appointed to different classes of service and the network treats the packets that belong to each class differently. The Differentiated Services Code Point (DSCP) is a field in the IP header that specifies the class of service each packet belongs to. The network classifies and marks the packets in order to provide to them differential per hop behavior (PHB) according to the class of service they belong to. The PHB is specified on all the network nodes and includes functions that implement resource allocation and packet drop policy. The packet scheduling mechanism determines the resource allocation. During the last years many such scheduling mechanisms have been proposed, such as Priority Queuing, Weighted Fair Queuing (WFQ), Modified Deficit Round Robin (MDRR) [5], [10]. The decision about the classification procedure and which packet scheduling mechanism will be used is up to the network administrator. This

decision should be taken at the design phase of the QoS service which will be offered on the network domain. Generally, a Bandwidth Broker should be customizable enough in order to allow changes on the QoS mechanisms.

A number of Bandwidth Broker implementations have appeared in the literature, and several architectures have been tested in actual scenarios [11], [12], [13] and in simulated environments [4], [14].

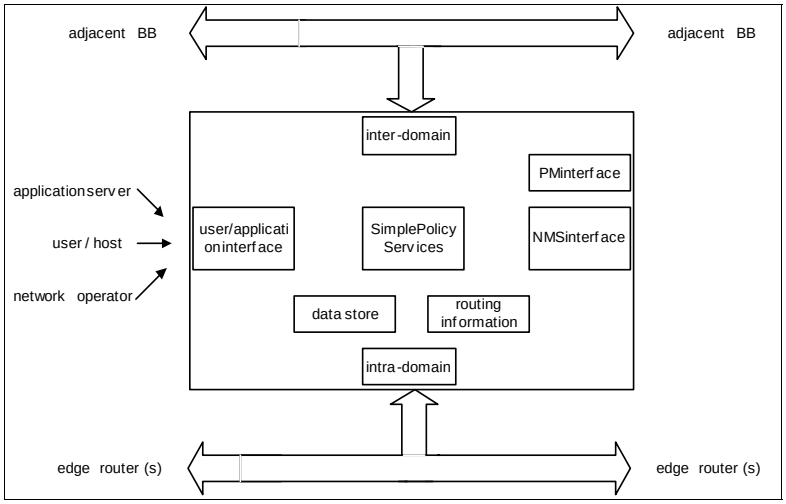


**Figure 2: Bandwidth Broker Architecture**

# 3 Bandwidth Broker Implementation

## 3.1 Simulation environment

Simulation has always been a valuable tool for experimentation and validation of models, architectures and mechanisms in the field of networking. It provides an easy way to test various solutions in order to evaluate their performance without the need of an actual network, which would have to be dedicated for experiments. In our case, a Bandwidth Broker has been implemented and tested on a simulation environment in order to evaluate its performance characteristics and its used mechanisms. The simulator that we used is the Network Simulator (NS-2), which is a free open-source simulator [8]. NS is available for anyone to download and it comes in many releases for various operating systems such as FreeBSD, Linux, SunOS, Solaris and Windows. Its current release is ns-2.27. NS-2 is a powerful simulation tool that can simulate many kinds of networks, such as mobile and satellite networks. A user can define arbitrary network topologies consisting of nodes and links and attach applications and queues on each node. A researcher using NS-2 can design new protocols, test their functionality and performance and compare them.

The simulator is written in C++ and uses OTcl as a command and configuration interface. NS-2 supports many network protocols such as TCP and UDP, many traffic sources such as FTP, Telnet, Web, CBR and VBR, queue management mechanisms, such as RED, DropTail and CBQ and routing algorithms such as Dijkstra and Bellman-Ford. NS-2 is an event driven simulator that takes tcl scripts as input and executes them. The output can have many forms, even graphical representation of the network. A common output can be files that thoroughly describe the traffic on a link or the condition of a queue. The output files can be additionally processed in order to calculate specific quantities such as the throughput of a traffic class.

During our work, we used the ns-allinone-2.26 release that contained some additional tools, apart from NS-2 and OTcl. The most important of these tools are:

- Nam. It is a tool that used to make graphical representations of the network topology and the network's operation
- x-graph. This tool is used to make graphical representations of some output data
- gt-itm. It is a mechanism of automatically producing network topologies

NS-2 is updated very often and many people support it either by fixing bugs or by writing new code that adds functionality to the simulator. Following this practise, the version that we used was extended with more DiffServ functionality as documented in [3], [9].

### 3.2 The implementation in NS simulator

The implementation of a Bandwidth Broker on NS-2 followed the classic architecture of a bandwidth broker as Figure 2 illustrates. In order to implement it correctly, it was necessary to make several changes and additions to the NS-2 structure and source code. An agent in NS-2 represents an endpoint where packets are consumed and constructed, using a specific protocol. The Bandwidth Broker that was implemented is based on two new agents, the Edge Bandwidth Broker and the Base Bandwidth Broker. More specifically, we created the classes BBedgeAgent and BBbaseAgent, derived from class Agent that implements the Edge Bandwidth Broker and the Base Bandwidth Broker. We also created two new packet types, BBB and BBE, which are used for the BB interfaces (to simulate the BB messages) and have a size of 64 bytes. BBbaseAgent creates BBB packets and consumes BBE packets created by the BBedgeAgent. BBedgeAgent creates BBE packets and consumes BBB packets created by the BBbaseAgent. In order to create a new packet type, it is necessary to define the header of the new packet. The header fields that we defined for the BBB and the BBE packets were the address of the sender of the RAR, the address of the other end node, the type of the packet (RAR or RAA), the amount of the requested bandwidth and the final answer the BaseBB sends to the sender (Negative or Positive). The total bandwidth that the BB manages on each link is determined by a new tcl instruction "set_bndw". The syntax is *"BBedgeAgent set_bndw node_id bandwidth"*. This instruction informs the BBedgeAgent for the bandwidth that the BB will manage on the link that exists between the node where the BBedgeAgent is running and its adjacent node with node-id node_id.

A BBedgeAgent, which represents a client (user / application), can send a RAR requesting guaranteed bandwidth between the node where it is running and another node with node-id node_id using the new tcl instruction "sendto". The syntax is *"BBedgeAgent sendto node_id bandwidth"*. The BBedgeAgent that exists on every node simulates a situation where a BB client is connected to a router on a real network. This agent operates as client that makes the communication with the base BB and updates its local router with the configuration modifications according to new admissions. In our case, this agent also stores data regarding the adjacent nodes of the node and communicates with the base BB every time the base BB needs this information. So, the architecture is partly distributed as some information is stored locally on every "client" and not centrally on the base BB.

### 3.3 The admission control algorithm

All this new functionality on NS-2 provides the capability to simulate a network that is managed by a Bandwidth Broker. The Base Bandwidth Broker Agent connects to every Edge Bandwidth Broker Agent. The system function is achieved with the use of the messages that are always sent either from an Edge Bandwidth Broker to a Base Bandwidth Broker or from a Base Bandwidth Broker to an Edge Bandwidth Broker. Two Edge Bandwidth Brokers never communicate by sending messages to each other, since the Base Bandwidth Broker always intervenes in the communication.

The system's operation begins when an Edge Bandwidth Broker makes a request asking guaranteed bandwidth of x bps from the node it is running to some other network node. If at that time the Base Bandwidth Broker serves another request, then it sends an answer informing the Edge Bandwidth Broker about its unavailability to serve the new request. Otherwise, the Base Bandwidth Broker begins to serve the request. Initially, it checks whether there is available bandwidth from the node where the Edge Bandwidth Broker that made the request runs, to the other end-node. Every Edge Bandwidth Broker maintains information about the available bandwidth between the node on which it is running and all the adjacent nodes. The bandwidth that is available to be managed by the Bandwidth Broker is specified on each edge bandwidth broker agent and can be a fraction of the total bandwidth. The Base Bandwidth Broker searches the routing tables to find the next hop $n_1$ from the node $n_0$ that made the request to the end-node $n_k$. Then, the Base Bandwidth Broker sends a query to the Edge Bandwidth Broker that runs on node $n_0$ asking if there is available bandwidth between the nodes $n_0$ and $n_1$. If the answer is affirmative, the Base Bandwidth Broker finds the next hop $n_2$ that lies in the path from node $n_1$ to node $n_k$ and sends a query to node $n_1$ asking if there is available bandwidth between the nodes $n_1$ and $n_2$. If all the answers are affirmative, this procedure continues until node $n_k$ is reached. This means that there exists available bandwidth from node $n_0$ to node $n_k$ and the Base Bandwidth Broker will send a positive answer to the Edge Bandwidth Broker that made the request so that node $n_0$ is notified that it is allowed to begin transmitting data to the backbone network. The procedure will be completed after the Base Bandwidth Broker sends to all the Edge Bandwidth Brokers that lie on the path $n_0,n_1,\ldots,n_k$, messages informing them to reduce by x bps the available bandwidth on

the links that constitute the path. In case one of the Edge Bandwidth Brokers sends a negative answer, because there is not sufficient bandwidth available on a link, the Base Bandwidth Broker sends a negative answer to the node that made the initial request and the procedure ends there.

Following the admission of a new request, the next most important point on the operation of the bandwidth broker is the resource allocation scheme that is used in order to provide the admitted guarantees.

### 3.4 The supported QoS service

The Bandwidth Broker provides to the users a QoS service that tries to provide bandwidth guarantees as well as minimum delay and jitter. This service is the IP Premium and is now 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 46 and 6 for admitted and downgraded packets. The policing is performed at the edge of the network and in the core routers only priority queuing is performed.

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. 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. We have also implemented the Modified Deficit Round Robin Scheduling Algorithm (MDRR) [6][15] and changed the whole queue management mechanism to enqueue packets based on DSCP. The QoS service, as it has been implemented, classifies the packets for each class that has been admitted by the bandwidth broker with DSCP value 46. Then, when the packets are inserted in the network, we apply strict token bucket policy in order to make sure that the transmitted rate agrees with the admitted rate. Next, on all the network nodes, the queue management mechanism is properly configured. The used queue management mechanism is a high priority queue on every node that is used for all the admitted traffic classes. Additionally, instead of priority queuing, the MDRR mechanism can be used. With this decision, the admitted traffic (for the QoS service) except from bandwidth guarantees, also benefits from low delay and jitter that is provided by the strict priority queues.

The role of the bandwidth broker in this stage is to reconfigure properly the backbone routers, every time a new request has been accepted. More specifically, when the admission control module responds with a positive answer, the Base Bandwidth Broker informs the edge bandwidth broker client that made the request and starts the procedure to configure the network devices. The Base Bandwidth Broker is aware of the routing information and determines the path and therefore the backbone routers that need to be configured. In this case, the Base Bandwidth Broker informs the necessary edge BB clients that operate on each backbone router. Next, the clients use the data structure where they save all the information about the links that they manage and their status to determine the information about the reserved amount of bandwidth. Moreover, they have been provided with a configuration template of the QoS service and therefore they create the appropriate configuration parameters for the queue management mechanism. The next step for each client is to apply the configuration parameters to the queue management mechanism (Priority Queuing).

Summarizing the procedure described above, upon receiving a request from an edge client, the Base Bandwidth Broker reacts according to the following pseudocode:

```
Get routing info from the source to the sink
Store the links that should be reconfigured in vector v
for all links in v
          determine the edge BB client that manages this link
          Determine the new configuration parameters
          send the new configuration parameters to the edge BB client
end for
```

Upon receiving a call from the base bandwidth broker, the edge bandwidth broker's function performs the following steps:

```
Retrieve from the data
structure the total reserved
bandwidth for the link

Apply the configuration
parameters to the QoS
mechanisms at the managed
routers
```

# 4 Experiments

The Bandwidth Broker, as it has been implemented on the NS-2 simulator and described in section 3, was tested for various network topologies and traffic loads. The scope of all the tests was initially to measure the performance of the implemented modules (algorithms) and afterwards to test a full scenario of the available bandwidth broker service in order to provide end to end QoS. The 4 network topologies that were used for the tests are shown in Figure 3. For every simulation test, the network was loaded with background traffic that was inserted with the cross connect method [3], [9].

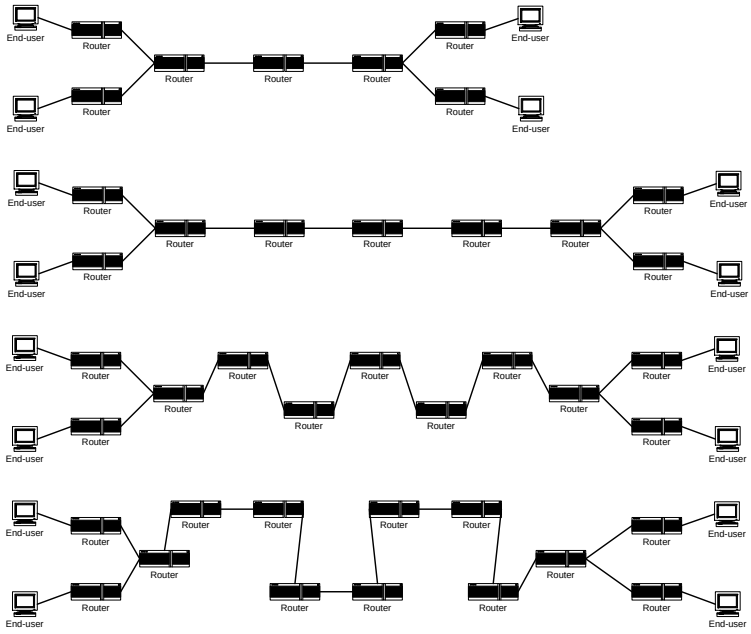


**Figure 3: The tested network topologies**

## 4.1 Performance Evaluation

Initially, we measured the packets that are exchanged every time a new request is submitted and the bandwidth broker admits it and are presented on Table 1. The number of the packets depends on the location of the Base Bandwidth Broker on the network. It is obvious from the admission control algorithm and the distributed nature of the implemented bandwidth broker that the number of the packets depends on the location of the Base Bandwidth Broker on the network. In case that the routing path of the new flow does not contain the router that the BB base is connected to, then the number of packets is larger.

| Topology | Packets sent by BB edge | Packets sent by BB base | Total packets |
|---|---|---|---|
| 1 | 6 | 14 | 20 |
| 2 | 8 | 20 | 28 |
| 3 | 10 | 26 | 36 |
| 4 | 12 | 32 | 44 |

**Table 1: exchanged packets for the above topologies**

Our next step was to evaluate the performance of the Bandwidth Broker and the overhead that it inserts in the network. As we presented above, the exchanged packets for each request and answer from the BB depend on the network topology and where the main Bandwidth Broker agent and the specific source and sink are located. The packets that are exchanged for the Bandwidth Broker purposes are quite small (64 bytes) and also are classified as high priority packets using the priority queues that are used for the admitted traffic. This choice has been done in order to avoid packet loss for BB communication if the network is congested. We also tried to measure the total time for the BB operation, which is the time from the moment that the source sends a request until the Base Bandwidth Broker agent answers positively or negatively. These measurements have been done for all the above network topologies (Figure 3) and for various time periods (every 50 sec) in order to make sure that they are independent of random situations of congestion.

| Measurement time (sec) | Topology 1 (sec) | Topology 2 (sec) | Topology 3 (sec) | Topology 4 (sec) |
|---|---|---|---|---|
| 50 | 0.106 | 0.170 | 0.267 | 0.331 |
| 100 | 0.105 | 0.174 | 0.243 | 0.307 |
| 150 | 0.107 | 0.165 | 0.244 | 0.300 |
| 200 | 0.103 | 0.171 | 0.246 | 0.306 |
| 250 | 0.108 | 0.169 | 0.235 | 0.310 |

**Table 2: Duration of bandwidth broker operation**

Studying the results in Table 2, we can conclude that the duration of the total operation to process a new request and answer positively or negatively depends on the network topology and in particular on the number of nodes and the length of the routing path. As the topology becomes larger, the duration of the total process increases. This result is also related with the implementation of the BB, because it keeps information on all BBEdge agents and requires communication of the BBBase Agent with all the BBEdge Agents on the routing path in order to admit a new request. In a new BB model where all the information will be stored locally on the BBBase Agent, this time can be reduced but then some of the advantages of the distributed model are lost. Therefore, the total duration of a BB operation can be reduced, but nevertheless the measured duration can not be considered as high, since it is less that 0.5 seconds.

### 4.2 Testing the whole Bandwidth broker service to provide end to end QoS

Having tested the BB operation, we then simulated the scenario were the backbone links are all 10Mbps and the BB manages 2Mbps on each link for QoS requests (as determined by the QoS service's dimensioning). The topology that was used is the first one in Figure 3. At this point, 2 sources requested 1Mbps each and were successfully admitted by the network as the total bandwidth was available. The throughput of the traffic that these 2 sources created is presented in Figure 4.

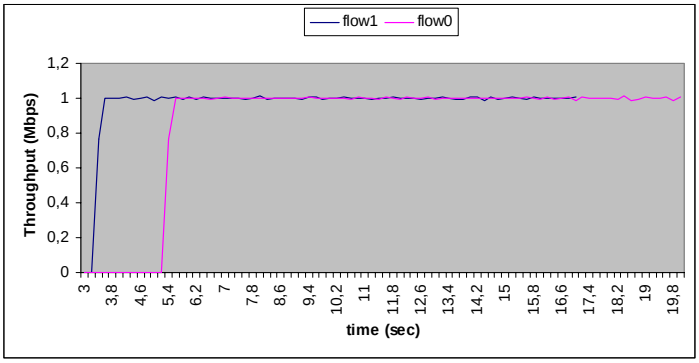


**Figure 4: Throughput of the admitted traffic by the BB**

According to this figure, the 2 sources created traffic with transmission rate exactly 1Mbps and the traffic was received by the recipient since it had crossed the network with high priority. This

result demonstrates the proper operation of the BB module, since it admitted the requested flows and it also succeeded in keeping these guarantees. The following experiments aim at simulating more complicated situations, such as the situation where the sources create traffic with higher rate than the rate admitted by the BB.

In this case, we simulated the above scenario with the difference that the sources request 1Mbps each, but as soon as the BB admits the requests, the sources start creating traffic with a rate of 2Mbps (which is a rate double than the one they requested). At this point, the operation of the policing mechanism of the IP Premium service is very crucial as it should limit the source's rate. The policing mechanism that the resource allocation scheme uses is token bucket policing and has been configured to drop the packets that exceed this profile. On the other hand, it could have been configured to treat the exceeded packets with best effort service, but in any case it is a policy decision regarding the operation of the network and we decided to simulate the stricter approach, where the exceeded packets are dropped. Several combinations of policing profiles were tested and are presented in Table 3. The policing profile consists of the committed rate and the burst size. At this point we should also mention that the policing profile is closely related to the transport protocol that the sources use and in our case the sources use the UDP protocol.

Figure 5 presents the results for the above policing profiles and Table 4 the average throughput that the flows experience using the above policing profiles.

| Policing profile | Token bucket rate | Token bucket depth |
|---|---|---|
| 1 | 1.1*admitted rate | 10*MTU |
| 2 | 1.1*admitted rate | 2*MTU |
| 3 | 1.1*admitted rate | 1*MTU |

**Table 3: The used policing profiles**

| Policing profile | Average throughput of flow 1 (Mbps) | Average throughput of flow 2 (Mbps) |
|---|---|---|
| 1 | 1.018 | 0.980 |
| 2 | 1.013 | 1.068 |
| 3 | 1.013 | 1.067 |

**Table 4: Average throughput of the flows for the 3 different policing profiles**

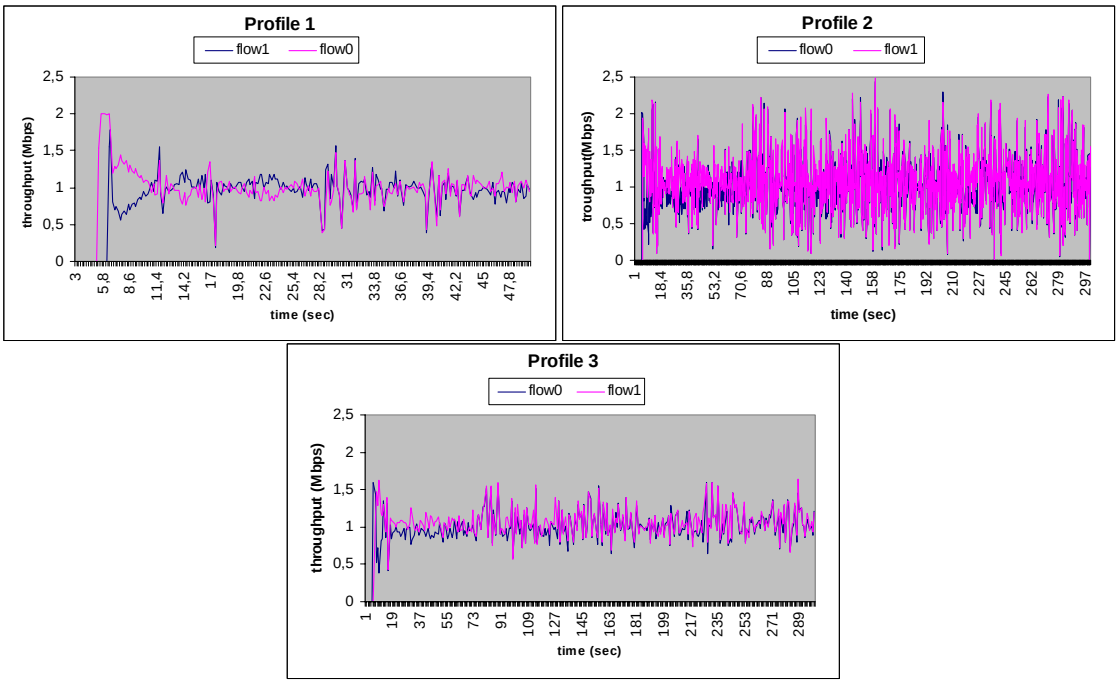


**Figure 5: The final throughput of the flows for the above policing profiles**

According to the results that are presented in Table 4 and in Figure 5, it is obvious that the policing profile should be as tight as possible in order to prevent the network from transmitting traffic that exceeds the admitted guarantees. The best choice seems to be to configure the committed rate slightly higher than the admitted (in the experiments it is 10% higher) and the burst size only a few packets (1-2 packets), as in this case the final throughput that the flows achieve is very close to the requested and also the variation of the throughput is normalized.

## 6 Conclusions - Future work

The main work that is presented in this paper is our effort to implement the basic functionality of a Bandwidth Broker on NS-2. Its operation is to manage a specific fraction of bandwidth on each link, receive requests and admit or reject them according to the bandwidth availability and the predefined agreements (SLAs). The Bandwidth Broker provides a QoS service to the admitted traffic using well-known DiffServ functionality. The tests that were described above demonstrate that the Bandwidth Broker operates well as it reacts properly for each request. Additionally, there was an investigation of the policing mechanism that should be used to avoid transmitting more guaranteed traffic than the admitted. Finally, we tried to evaluate the performance of the Bandwidth Broker by measuring the exchanged packets as well as the average processing time of a new request on the Bandwidth Broker, which is quite small.

All the tests that were performed also indicated some points that need further research and investigation. Our first goal is to investigate alternative admission control algorithms. Also, we plan to implement a more centralized second version of the Bandwidth Broker where all the information will be stored on the main BB agent and the admission control will be centrally performed. The edge BB agents will inform the base BB for new entries (new nodes on the network) and will also periodically update the information on the base BB agent. This version will be tested and compared with the current distributed version in order to find the appropriate trade-off between distributed and central model. Finally, we plan to implement the communication protocol between two adjacent Bandwidth Brokers and simulate the scenario of interconnected domains that are managed by independent Bandwidth Brokers.

## References


1. RFC 2475 “An Architecture for Differentiated Services”, S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, December 1998
2. RFC 2905 “AAA Authorization Application Examples”, J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, D. Spence, August 2000
3. C. Bouras, D. Primpas, A. Sevasti, A. Varnavas “Enhancing the DiffServ architecture of a simulation environment”, 6th IEEE International Workshop on Distributed Simulation and Real Time Applications, Fort Worth, Texas, USA, October 11 – 13, 2002
4. Olov Schelén, Andreas Nilsson, Joakim Norrgård, Stephen Pink “Performance of QoS Agents for Provisioning Network Resources”. In Proceedings of IFIP Seventh International Workshop on Quality of Service (IWQoS'99), London, UK, June 1999
5. S. Vegesna, ‘IP Quality of Service: the complete resource for understanding and deploying IP quality of service for Cisco networks’, Cisco Press, 2001
6. C Bouras, M. Campanella, M. Przybylski, A. Sevasti “QoS and SLA aspects across multiple management domains: The SEQUIN approach” (http://www.elsevier.com/locate/future). Future Generation Computer Systems 19 (2003), pp 313-326
7. G. Fankhauser, D. Schweikert, and B. Plattner, "Service Level Agreement Trading for the Differentiated Services Architecture," Tech. Rep. 59, TIK, 1999
8. http://www.isi.edu/nsnam/ns/ns-build.html
9. http://ouranos.ceid.upatras.gr/diffserv/start.htm
10. http://www.cisco.com
11. Manzoor Hashmani and Mikio Yoshida "ENICOM's Bandwidth Broker", Saint 2001 Workshops, pp 213-220, Jan 8-12, 2001, San Diego, USA
12. QBone Bandwidth Broker Architecture, http://qbone.internet2.edu/bb/bboutline2.html
13. J. Ogawa, A. Terzis, S. Tsui, L. Wang, L. Zhang. “A Prototype Implementation of the Two-Tier Architecture for Differentiated Services”, RTAS99 Vancouver, Canada
14. 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, August 30 - September 1 2004