Performance of a Bandwidth Broker for DiffServ Networks

Günther Stattenberger and Torsten Braun

Institute of Computer Science and Applied Mathematics,
University of Bern,
Neubrückstr. 10,
3012 Bern,
Switzerland
{stattenb, braun}@iam.unibe.ch
http://www.unibe.ch/~rvs

Abstract. The need of a powerful tool for the management of large backbone networks is growing. Furthermore, new applications and user behavior require the ability to quickly adapt the network configuration to a dynamically changing environment. We present an architecture and an implementation of a bandwidth broker for DiffServ network management. The performance evaluation shows, that our implementation is able to serve as the central management station for a network containing several hundred nodes while still providing a short flow setup time. Further improvements of the current implementation are discussed, too.

1 Introduction

Managing a backbone network of even a medium size Internet Service Provider (ISP) is a difficult and time-consuming task. The sheer number of routers and the heterogeneity of the network (i.e. the routers are bought from different vendors and therefore have different command line interfaces for configuration) complicate the configuration of the network and this often results in a rather static configuration of the backbone routers. However, in modern communication networks dynamic and frequent changes in the settings of some of the backbone routers are necessary to fulfill the requirements (e.g. bandwidth reservations) of customers (e.g. mobile users). A bandwidth broker (BB) can support the network administrator of an ISP network in various ways: first the heterogeneity of the network can be hidden behind a uniform configuration interface. Furthermore, the configuration of the routers can be automated and completely delegated to the broker. The BB offers a small set of possible contracts to the customers (so-called Service Level Agreements, SLAs) and takes care of the correct completion of those contracts. The users can therefore dynamically set up and release resource reservations at high rates, which is very convenient and satisfactory for the customer.

The difficulties of deploying an interdomain service in the internet are listed and discussed in [11]. For example, a big barrier are the necessary upgrades for providers, since *all* ingress interfaces of the whole domain must police. In addition, the current lack of router support poses a big problem. Serious concerns about operational cost and

complexity have to be considered, too. Some of those problems are addressed by existing architectures of bandwidth brokers [9, 10]. All of them try to reduce the complexity of the configuration procedure. The different approaches mainly follow the design of the two-tiers managment architecture presented in [8].

In this paper we present an architecture and an implementation of a bandwidth broker capable to manage and configure even large-scale networks with an acceptable speed. This broker introduces a novel object-oriented way of interconnecting the management and configuration layer of the common bandwidth broker architecture. This interconnection is built on top of a QoS Management API [15], that can be used to build a virtual network representation that combines a topology database and the knowledge of the amount of traffic reserved at each router interface. The features of the broker include a generic communication interface for user - broker communication as well as for broker - broker communication and a policy database providing methods for subnet-based admission control. Altogether we are able to show the ability of our architecture to perform the necessary actions in several separated ISP networks in order to provide end-to-end service guarantees for the user.

The rest of this paper is organized as follows: Section 2 discusses some bandwidth broker implementations of other authors, in Section 3 we explain our bandwidth broker architecture using a single-ISP broker as an example. Section 4 evaluates the memory consumption and the reservation setup speed of this architecture. Section 5 discusses several enhancements of the basic architecture and their possible effects on the overall performance.

2 Related Work

Several bandwidth brokers have been designed and developed through the last years since having been introduced in [8]. This section will focus on the differences of some of the most referenced bandwidth broker implementations [4, 6, 7, 10, 17], and discuss some of their characteristics and drawbacks. While not presenting a broker implementation for DiffServ [6] is included in this discussion for more general aspects of network management.

When developing a bandwidth broker several design choices are quite canonical, therefore the different architectures and implementations differ in several details only: A flow database that contains the flow requests from the users is available in each implementation, only the interface to this database is different. For example, [4] uses a set of configuration commands usable by a client host, while [17] uses a web interface to the database that only a network administrator is allowed to use. The second database which occurs in all implementations is a policy database. This is the database deciding whether a flow request can be served or not. COPS (Common Open Policy Server) is chosen in several implementations [7, 17] to handle the communication between the bandwidth broker (Policy Decision Point (PDP)) and the routers (Policy Enforcement Point (PEP)) but also alternative approaches are being proposed, such as SNMP in [10] or TCP sockets and telnet in [6, 4]. If a detailed flow description is documented, this also contains the same entries throughout several different implementations: source and

destination addresses as well as ports, protocol ID, start time, duration and service level information (such as bandwidth etc.).

One point that is missing in many documentations about implementations is the discussion about topology knowledge. One implementation [10] proposes to introduce special route discovery signaling using the IP record route option, but other implementations neglect this topic. A topology database is used in [6], but the authors fail to mention, how this database is built and kept up to date.

Regarding the performance of the different implementations the evaluation tests that are presented in the publications merely represent a functionality test not exceeding the setup of DiffServ reservations at one single router. No tests are shown how many routers can be configured by the implementation simultaneousely, and how much time it takes for a flow reservation request to be processed.

3 The Bandwidth Broker Architecture and Implementation

Our bandwidth broker implementation is based on the implementation of the QoS Management framework [15] for a Linux DiffServ (DS) router. The framework roughly consists of three abstract C++ classes, that define a generic interface for the three basic building blocks of a DiffServ network: a router, an interface at a router and a traffic conditioner at the interface. The implementation of this framework for the Linux DS router implements specific C++ classes derived from those abstract base classes to support the specific needs of the Linux Router (e.g. command line syntax, routing table format ...). Different router hardware is supported by deriving special child classes for each type of router.

The BB uses those three classes to build a representation of the underlying network topology. It can configure the router hardware by using common configuration commands (such as `add_flow`). Those generic commands are translated into the configuration scripts by the corresponding `Router` instance. This solves the problem of supporting various kinds of router hardware by adding a software layer flexible enough to deal with various routers by using different derivations of a common base class. The base class provides an interface identical to all routers thus hiding the differences from the user.

The API additionally provides the necessary functionality to keep track of the amount of bandwidth reserved for a specific DiffServ class at each interface of each router. Figure 1 shows the architecture in more detail: Our bandwidth broker contains a flow database, that holds all registered flows. This is one of the two databases that are always present in a broker implementation. The second one, the policy database is not used in this simple scenario but will be added in the multi-ISP scenario (see Figure5). As a new feature, our architecture has a virtual representation of the network to be managed. This representation contains all routing tables of the routers from the network, thus providing a topology database. This database will use a large amount of memory, but the forwarding path of a flow can be obtained very quickly without additional signaling. In addition, this virtual network offers the common router configuration interface of the management framework to the broker's management software, making it easier for new management implementations to be deployed. Our implementation keeps track of the

link utilisation as well as the reserved bandwidth per link. Those parameters are stored in two tables, enabling the broker to perform basic admission control based on the link utilisation. Finally a user interface offers a set of generic flow management commands (add_flow, delete_flow, change_flow, list_flows) to the customer.

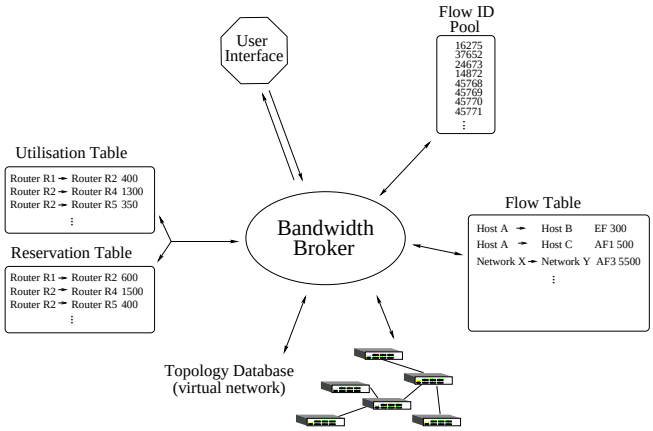

Fig. 1. The bandwidth broker architecture for an isolated network

During the initialization the broker automatically generates a network representation. This can be done either by reading a local topology database from the hard disk, or by broadcasting a broker advertisement message. Each router has to reply to this advertisement message giving its host name and its router type. We chose this approach because of its stability against changes in the topology, although we had to accept the resulting need of a router daemon running on each router. Since we anyhow depend on a router daemon to perform the configuration commands from the bandwidth broker, this is not a big drawback.

After this step the bandwidth broker fetches all routing tables from the network. It now has the full knowledge of the topology on the IP layer, which is sufficient for our task (i.e. configuring DiffServ flows). In addition, all necessary traffic conditioners needed to support DiffServ are already set up and a default configuration is loaded. This default configuration simply consists of empty DiffServ reservations at each interface.

The bandwidth broker can now load an initial DiffServ configuration to the network. This is perhaps necessary, if there are contracts with adjacent bandwidth brokers concerning DiffServ traffic entering or leaving the domain from outside. Additionally a network administrator can choose to reserve a certain amount of bandwidth for DiffServ traffic in advance in order to reduce the total configuration overhead in the backbone. The programmer of a bandwidth broker should generally take care of reducing the amount of reconfigurations of a backbone router. Overprovisioning, i.e. reserving more bandwidth than it is actually needed by the running applications is one way to relieve the backbone routers [2].

Finally, the bandwidth broker is ready to accept incoming reservation requests from the customers. The protocol that is used for the communication between the bandwidth broker and the user is described in [14]. This protocol allows the user to specify a flow (i.e. source and destination) together with the required bandwidth and excess bandwidth as well as some flags indicating the additional requirements of the flow (e.g. realtime traffic). For each request the BB sequentially queries the routing tables from the source to the destination and thus finds all routers that potentially might need to be reconfigured (the result is equivalent to a `traceroute` from source to destination). This approach results in a delay thet is proportional to the diameter of the topology (i.e. the average number of hops between two arbitrary nodes). As an alternative, [10] proposes to use the IP *record route* option and ICMP messages to discover the route of a given flow. This has the advantage of providing up-to-date information, but adds an additional delay to the setup procedure. Furthermore, the problem of route breakdowns and the resulting reconfigurations in order to provide guaranteed QoS to all registered flows is not addressed. This topic is addressen in Section 5.3, where we show, how a central topology database could be kept up to date quite easily.

Following this, an appropriate DiffServ Codepoint (DSCP) based on the requirements of the user is chosen. Now the ingress router can be configured. Note that in a DiffServ network the ingress router has always to be reconfigured, since it is responsible for shaping the traffic based on the exact flow description. The bandwidth broker can check, if it is necessary to perform reconfigurations at the core and egress routers (e.g. adjust the EF rate limiter in a core router [1]). This is based on the brokers values of the total amount of traffic and the reservations at each interface. Usually there is more bandwidth reserved than actually used due to over-provisioning, and no reconfigurations have to be done. The broker only has to update its bandwidth usage variables for the interfaces. This approach speeds up the flow setup significantly.

In this rather simple scenario of an isolated network only a very basic form of admission control is performed: at each router we check, if the newly allocated amount of bandwidth for the service classes does not exceed a pre-configured threshold. If so the flow is rejected. It is guaranteed, that a certain amount of bandwidth is avaliable for best effort traffic in every case. A more elaborate form of admission control based on a subnet-indexed policy database will briefly be discussed in Section 5.1.

Finally the BB stores the flow request in its global flow database. This database assigns a unique flow ID to each new flow and passes this flow ID to the user. All future operations depending on this specific flow setup are referenced by this ID.

4 Performance Evaluation

The performance evaluation of our architecture shows, how fast the bandwidth broker can handle flow requests when managing a reasonable large network. In addition, we were interested in the memory consumption of the bandwidth broker, which we expect to be quite large because of the central topology and flow databases. All measurements were performed on a dual-processor AMD Athlon MP 2000+ Linux PC with 2 GB main memory. For the evaluation several test topologies have been created using the tiers [3] program. This program randomly generates a topology consisting of a single WAN with

several MANs and several LANs per MAN. The number of routers per network can be chosen freely. We generated topologies containing 157, 208 305, 535, 710, 928, and 1010 nodes.

For the evaluation it is necessary to run a very large number of configuration daemons in parallel. Using a dedicated Linux router per daemon would be too much a financial and organisational overhead, since the load (in terms of CPU time) such a daemon creates on a PC is minimal — its only job is the execution of configuration commands, that are not computationally intensive. Therefore, we were able to run a number of router configuration daemons (about 200) on a single PC without creating too much load on the CPU. This does not at all diminish the amount of work to be done by the bandwidth broker since the access link is by far fast enough to handle all flow requests. Most likely, the bandwidth broker would be even faster than shown in our results, since the configuration requests are handled one after another on a single machine and are not processed in parallel on multiple machines. The only restriction we had to make was to disable the changes in the routers forwarding path: one can understand easily that a large amount of configuration daemons changing the settings of the Linux router simultaneously would result in a huge confusion and cause the router to crash. However each router configuration daemon program still parses each flow request and creates the necessary configuration scripts that would be needed to configure the router. Thus as much as possible of the computation expense is preserved.

The load on the bandwidth broker is created by requesting many flow reservations in a short period of time. Those reservation requests are generated by a small application that sends a request to the bandwidth broker, waits for the acknowledgement from the broker and immediately afterwards sends another request. By sending several hundreds of reservation requests from this program we can easily calculate the time the bandwidth broker needs for setting up a flow.

4.1 Results

For calculating the memory consumption, we used 7 tiers topologies. Using the Bellman - Ford distance vector (DV) algorithm [16] routing tables have been generated for each node in the networks and saved to the harddisk. With this algorithm each router has a routing table containing an entry for each router in the network. For each node of those topologies a router configuration daemon program has been executed. This daemon reads the routing table from the harddisk and passes it to the BB. The broker collects all routing tables from the daemons and builds the topology database.

The memory consumption of a centralized application is always a critical issue. Our bandwidth broker relies on two large databases — the topology database containing the routing tables and the flow database. The flow database is not very critical because only a very small amount of data has to be stored per flow (ca. 50 - 60 byte), and the memory consumption of this database grows linear with the number of established flows requests. However, in our scenarios the total size of all routing tables will grow quadratically with the number of nodes in the topology. This is an effect of the DV algorithm, that creates an routing table entry for each router of the network. Nevertheless we can estimate, how much memory a router instance consumes when its routing table contains a reasonable number of entries (ca. 10000 - 30000). This is approximately the size of

the routing table of a backbone router in a large ISP network. A simple quadratic interpolation shows, that the amount of memory per routing table containing 30000 entries is about 15 MB. This means that with our workstation equipped with 2 GB memory we could manage more than 100 backbone routers.

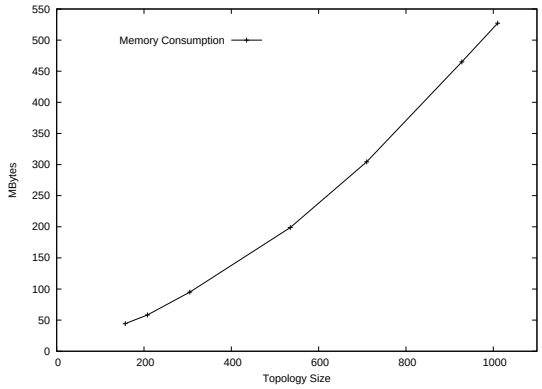

Fig. 2. Memory consumption of the bandwidth broker

Figures 3 and 4 show the flow reservation speed of the bandwidth broker. Each flow request reserved an amount of 100 kbit/s bandwidth between randomly chosen source and destination nodes. However, the amount of bandwidth reserved has absolutely no influence on the setup speed of the bandwidth broker. The time measurements were performed using the UNIX `time` command. This command measures the total (wall-clock) time a command needs for execution, as well as the "user time", i.e. the time that is spent in the program code, and the "system time", the time that was spent by the system, e.g. for performing I/O calls. The results show the total time the client application spent performing a given number of flow requests.

Topology Size	Best Case Speed (Flows/s)	Worst Case Speed (Flows/s)
710	1546.3	970.9
928	1836.2	1280.4
1010	1490.9	930.2

Table 1. Flow setup speed

In Figure 3 the performance of the bandwidth broker under optimal circumstances is presented. This is, when the bandwitdh broker has already allocated enough bandwidth on the whole forwarding path (by overprovisioning) so that only the ingress router has

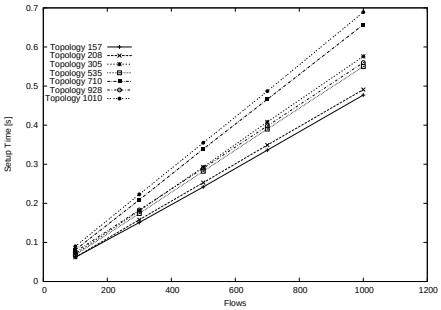

Fig. 3. Flow reservation time of the bandwidth broker (best case)

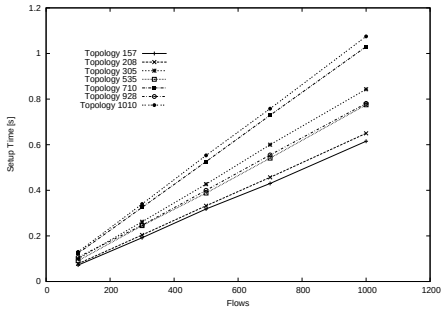

Fig. 4. Flow reservation time of the bandwidth broker (worst case)

to be reconfigured. Each graph represents one topology and shows the time that was needed to set up 0 – 1000 flows in seconds. In Table 1 we show the speed of the bandwidth broker measured in flow setups per seconds, measured over 10000 Flows setups.

In both results we can see, that the speed (i.e. the slope of the graph) does not directly depend on the topology size. The important parameter is in fact the diameter of the topology. Due to the randomness of the topology generation the diameters differ from e.g. 20.5 hops for the 710 nodes topology to 12.2 hops for the 928 nodes topology.

Figure 4 and Table 1 show the performance of the bandwidth broker under worst conditions: now the bandwidth broker had to reconfigure every router along the path. We had to force this situation by disabling the overprovisioning algorithm in the bandwidth broker. Again we can see the aforementioned dependency on the diameter of the topology. The total performance of the broker is however not bad considering the fact that compared to the best case scenario we have to reconfigure in average 12 – 20 routers per flow. But reconfiguring a core router is much easier than an ingress router, since only a single variable has to be changed (the limit of the aggregated bandwidth), while at the ingress router a new traffic conditioner has to be inserted and configured.

5 Further Improvements

The results presented in the previous section show, that our architecture is capable of configuring a backbone network at a reasonable high speed. Nevertheless, there are still some possibilities for optimization. First of all, the computing power (CPU speed and I/O bandwidth) of the bandwidth broker host has a big impact on the overall performance. Further results on this topic can be found in [13].

5.1 Hierarchical Bandwidth Brokers

Dividing an ISP domain into independently managed subnetworks will distribute the requests of the users to different bandwidth brokers, each capable of performing a certain amount of flow setup procedures per second. Therefore the overall capacity of manageable user requests can be extended. Such a hierarchy of bandwidth brokers has already

been presented in [18]. The authors can show, that this can reduce the processing overhead for a flow reservation.

Such an enhanced version of the bandwidth broker can also be used for a multi-ISP scenario, when we cannot assume, that all routers belong to the same domain. This is a much more realistic scenario, and has multiple applications, e.g. in Mobile IP management.

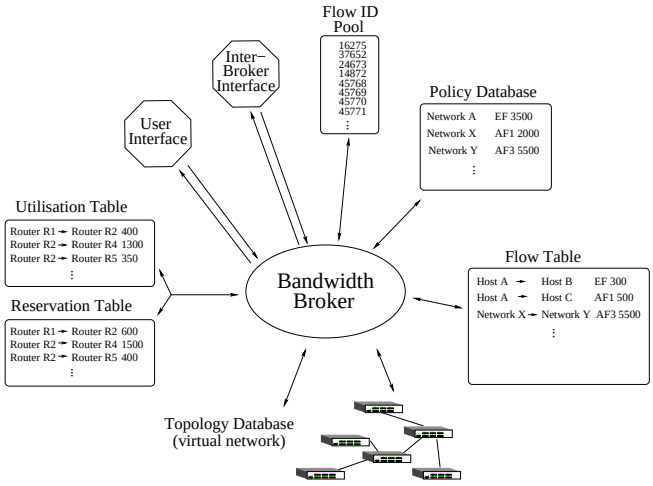

Fig. 5. The bandwidth broker architecture for multi-ISP scenarios

In [9] a hierarchy of brokers (called Resource Control Points) is presented, too. The authors mention, that dividing the network into parts, each managed by a single broker is not trivial and needs the knowledge of the topology as well as information about the expected SLAs. Therefore this topic is still under development.

Figure 5 shows the necessary extensions for a BB capable to communicate with other brokers: The protocol between the brokers as used by the inter-broker communication interface can be the same as in the user-broker communication, as it has been described in [14]. Using this protocol the brokers can reserve aggregations of flows by masking the source / destination address just like any user can reserve a flow at the broker. In addition, a policy database for restricting the amount of traffic from / to other domains is necessary. This policy database contains the information about the amount of traffic the specific subnets are allowed to send. This rather coarse access restriction can surely be improved, but it is e.g. sufficient to limit the number of users on a limited access network (e.g. a wireless access point). Based on this policy database we can provide admission control on the subnet of sender / receiver address of a flow.

If an ISP wants to split its network into several independent management domains, the separation will most likely be done on a IP subnet basis: Usually nodes with the same subnet mask are topologically close and therefore aggregation can be performed

much easier. In addition, the separation of the ISP network into several IP subnets automatically uniquely maps a bandwidth broker to the nodes of the subnet: As mentioned in Section 3 each broker can send a broadcast message for detecting the network topology. Due to the correlation of broadcast addresses and subnet masks each broker only gets a reply from nodes within its management domain. Any flow reservation request containing a source or destination IP address unknown to the broker denotes a flow involving several domains. The broker will then forward the flow request to the bandwidth broker the flow will cross next. For getting the IP address of the bandwidth broker of a foreign network, an additional service has to be established at the border routers.

5.2 Parallelizing the Broker

As already mentioned in Section 4 the flow requests are currently handled sequentially. Since configuring the network implies the bandwidth broker waiting a significant amount of time, it would be better to handle the user requests in parallel threads. Since it is not very likely, that many consecutive flow requests require the reconfiguration of one single backbone router, the probability of decoupling is quite high and a significant performance boost will result.

5.3 Topology Changes

Throughout this paper we assumed the topology to remain static after the startup of the bandwidth broker. This allowed us to simplify the design of the topology and the reservation databases. However, this assumption may not be valid in a large network over a longer time interval. Although the routes to popular destinations seem to be reasonable stable — despite the large number of BGP updates [12] — it might happen that a route changes while a reservation is still relying on the forwarding given by this route. We now want to shortly discuss possible solutions for this problem, which is still not solved in any bandwidth broker implementation.

The problem of routing changes is the aggregation of flows in the core routers: the core routers do not store per-flow information because of scalability reasons but only the overall sum of bandwidth is kept. If a link goes down, the underlying routing protocol might change the routing in a way, that one part of the flows is now routed via a interface, but another part via another interface of this router (and / or might not cross this router at all). Since the core router only has information about the aggregate sum of bandwidth we cannot know how to divide this sum into parts needed for the new routes.

One possible solution would be the allocation of the total sum of bandwidth on the new outgoing interfaces. This of course would result in a large overprovisioning and waste of bandwidth and is therefore not desirable. We propose another solution that breakes the rule of only storing aggregate information in the core routers in a way, that the bandwidth broker maintains a reservation value not only for each interface of a core router but for each routing table entry (see Table 2).

With this additional information we can correctly change the topology and reservation database in case of routing changes: Let us assume, the link of Interface 1 breaks, and the routing protocol provides us with the following new routes: destinations Dest_1 and Dest_2 are now routed via interface Ifc_2, while Dest_4 is routed via Ifc_3. The new

Destination	Gateway	Interface	Reservation
$Dest_1$	$Gatew_1$	Ifc_1	500
$Dest_2$	$Gatew_1$	Ifc_1	300
$Dest_3$	$Gatew_2$	Ifc_2	700
$Dest_4$	$Gatew_1$	Ifc_1	700
$Dest_5$	$Gatew_3$	Ifc_3	500
$Dest_6$	$Gatew_2$	Ifc_2	200

Table 2. Example for the new routing table format

Destination	Gateway	Interface	Reservation
$Dest_1$	$Gatew_2$	Ifc_2	500
$Dest_2$	$Gatew_2$	Ifc_2	300
$Dest_3$	$Gatew_2$	Ifc_2	700
$Dest_4$	$Gatew_3$	Ifc_3	700
$Dest_5$	$Gatew_3$	Ifc_3	500
$Dest_6$	$Gatew_2$	Ifc_2	200

Table 3. The routing table after the topology changes

routing table now looks like Table 3. The bandwidth newly to be configured at the interfaces can easily be computed from the routing table: Ifc_2 needs 1700 units, while Ifc_3 will be configured 1200 units.

Deletion of a routing table entry is also quite simple: we will only have to check the new routing table, over which gateway the addresses of the deleted subnetwork will now be routed. The only thing we have to do now is to add the bandwitdh used for the deleted network to the reservation entry of this gateway.

6 Summary and Conclusion

In this paper we have presented the design of a centralised bandwidth broker for Differentiated Services networks. Our implementation is based on a generic QoS management framework providing the necessary functionality. The performance evaluations of our implementation focused on two critical factors of possible bottlenecks: the memory consumption and the flow processing time of the central bandwidth broker. We could show, that the amount of main memory nowadays available at a workstation is sufficient to hold a large topology database. The flow setup time presented here was limited by the CPU speed and memory bandwidth of the hardware used during the measurements. Besides using faster hardware additional enhancements of the current architecture have been proposed, such as a hierarchical bandwidth broker architecture or parallelising of the flow request processing. Hierarchical architectures have already proven successful in increasing the performance of a bandwidth broker and also a parallel processing will help to make the broker perform even better.

7 Acknowledgement

The work described in this paper is part of the work done in the project Mobile IP Telephony (MIPTel) funded by the Swiss National Science Foundation (Project No. 2100-057077.99/1).

References

1. B. Davie, A. Charny, J.C.R. Bennet, K. Benson, J.Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis. RFC 3246: An Expedited Forwarding PHB (Per-Hop Behavior), March 2002. Obsoletes RFC2598 [5].

2. G. Dermler, M. Günter, T. Braun, and B. Stiller. Towards a Scalable System for Per-Flow Charging in the Internet. In B. Bodnar, editor, *Applied Telecommunication Symposium*, volume 32 of *Ariel Sharon Simulation Series*, 2000.
3. Matthew Doar. Tiers topology generator. http://www.geocities.com/ ResearchTriangle/ 3867/ sourcecode.html.
4. Bandwidth Broker Implementation. www.ittc.ukans.edu/ $\sim$ kdrao/ BB/ bbreport.html.
5. V. Jacobson, K. Nichols, and K. Poduri. RFC 2598: An Expedited Forwarding PHB, June 1999. Obsoleted by RFC3246 [1].
6. I. Khalil and T. Braun. Implementation of a Bandwidth Broker for Dynamic End-to-End Resource Reservation in Outsourced Virtual Private Networks. In *The Conference on Leading Edge and Practical Computer Networking*, November 2000.
7. P. Kivimäki. Policy Based Networks & Bandwidth Broker. www.atm.tut.fi/ workshop01/ workshop01-bb.pdf.
8. K. Nichols, V. Jacobson, and L. Zhang. RFC 2638: A Two-bit Differentiated Services Architecture for the Internet, July 1999.
9. G. Politis, P. Sampatakos, and I. Venieris. Design of a Multi-Layer Bandwidth Broker Architecture. In Sathya Rao and Kaare Ingar Sletta, editors, *Next Generation Networks — Networks and Services for the Information Society*, volume 1938 of *Lecture Notes in Computer Science*, October 2000.
10. O. Pop, T. Mahr, T. Dreilinger, and R. Szabo. Vendor-Independent Bandwidth Broker Architecture for DiffServ Networks. In *Proceedings of the IEEE International Conference on Telecommunications*, 2001.
11. QBone Premium Service. http://qbone.internet2.edu/premium/, March 2002.
12. J. Rexford, J. Wang, Z. Xiao, and Y. Zhang. BGP Routing Stability of Popular Destinations. In *Proceedings of the Internet Measurement Workshop 2002*, November 2002.
13. G. Stattenberger. *Scalable Quality of Service Support for Mobile IP Users*. PhD thesis, University of Bern, December 2002.
14. G. Stattenberger and T. Braun. QoS Provisioning for Mobile IP Users. In H. Afifi and D. Zeghlache, editors, *Conference on Applications and Services in Wireless Networks, ASW 2001*, Paris, July 2001.
15. G. Stattenberger, T. Braun, and M. Brunner. A Platform - Independent API for Quality of Service Management. In *Proceedings of the IEEE Workshop on High Performance Switching and Routing*, May 2001.
16. A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 3^{rd} edition, 1996.
17. A. Terzis, J. Ogawa, S. Tsui, L. Wang, and L. Zhang. A Prototype Implementation of the Two-Tier Architecture for Differentiated Services. In *Proceedings of the Fifth IEEE Real-Time Technology and Applications Symposium*, June 1999.
18. Zhi-Li Zhang, Z. Duan, and Y. T. Hou. On Scalable Network Resource Management Using Bandwidth Brokers. In *Proceedings of the Network Operations and Management Symposium*, April 2002.