Layered Network QoS Signalling - Motivation, Implementation & Measurements

Jens Schmitt[1], Martin Karsten[1], and Ralf Steinmetz[1,2]
[1] Industrial Process and System Communications, Darmstadt University of Technology, Germany
[1] German National Research Center for Information Technology, GMD IPSI, Darmstadt, Germany
Email: {Jens.Schmitt,Martin.Karsten,Ralf.Steinmetz}@KOM.tu-darmstadt.de

Abstract

The support of a single signalling protocol for all components on the data path of a QoS-based transmission cannot necessarily be assumed. In this paper, we investigate the issues surrounding hierarchically layered network QoS signalling configurations, as e.g. can be found in RSVP over ATM signalling. After introducing and discussing these issues we conclude that many of the decisions involved require understanding the performance characteristics of such layered signalling configurations. We therefore describe the implementation of an RSVP/ATM edge device, which we then use to conduct measurements in a configuration that involves in effect three layered signalling protocols: RSVP and ATM's UNI and PNNI. Using these measurements we review the issues in layering signalling protocols and reinforce design decisions being taken for the edge device mediating between the different mechanisms of the two network QoS architectures.

Keywords*: Network QoS, Signalling Protocols, RSVP, ATM, Overlay Networks.*

1. Introduction

The Internet has respected heterogeneity in underlying network technology from its inception as a matter of fact [1]. This was certainly one of the main reasons for it to become the global information infrastructure it is nowadays, as it allowed for a smooth evolution in time and space and continues to do so. The next step in the maturation of the Internet is towards a commercialization based on differentiation of the provided network service induced by different application requirements and user demands - the concept of *Quality of Service (QoS)*. At the moment there are manifold techniques for improving the plain best-effort IP towards the support of QoS, as e.g. RSVP (Resource reSerVation Protocol) [2], IntServ (Integrated Services) [3], DiffServ (Differentiated Services) [4], MPLS (Multiprotocol Label Switching) [5], ATM (Asynchronous Transfer Mode), etc. To us it seems very likely that none of these or any future QoS technologies will be able to oust the others, as there are always design trade-offs to make and therefore they all have different strengths and weaknesses. Hence we argue that the recognition of heterogeneity with respect to QoS technology used in the Internet is inevitable, see also [6] for a similar discussion.

In this paper we study the issues of QoS signalling in layered network systems. We do so by looking closely at a concrete example: RSVP-signalled IP flows over large-scale ATM subnetworks using UNI (User Network Interface) for access and PNNI (Private Network Node Interface) for inter-switch signalling. This should be seen as one particular instance of (triply) layered network QoS signalling that will allow us to investigate the issues in such configurations. Another example of layered signalling could be the interaction of RSVP/IntServ with a DiffServ subnetwork which employs a so-called Bandwidth Broker [7] to set up dynamic, and therefore signalled SLAs (Service Level Agreements).

As we argue that the issues of such layered network QoS signalling configurations involve understanding functional as well as performance characteristics of such layered signalling systems, we describe the design and implementation of an edge device that mediates between the RSVP/IntServ and ATM QoS architectures. This edge device is then used to conduct measurements of the performance characteristics of layered QoS signalling in a fairly realistic IP/ATM testbed. We emphasize on measurements of this particular configuration but argue that the considerations that have to be taken into account for such layered signalling apply equally well to other configurations.

2. Layered Network Signalling - Model&Issues

2.1 Conceptual Model

We assume an overlaid network configuration, where network elements on the same level communicate with each other by the use of signalling protocols for the purpose of conveying QoS. However, to do so they also trigger lower-level signalling for intermediate devices to coordinate QoS

on one section of the higher level QoS path. This may take place recursively as shown in Figure 1.

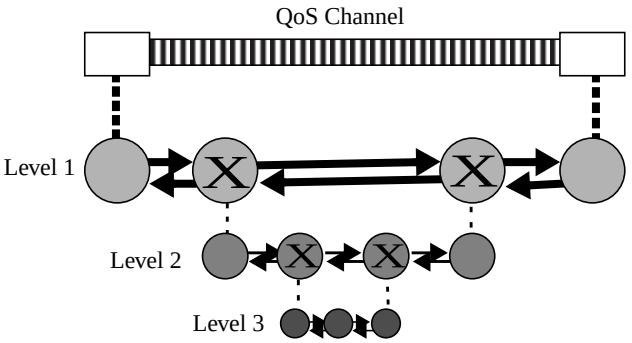


Figure 1: Layered Signalling Model - Concatenation.

In effect a uni- or bidirectional QoS channel between two or several instances is established. Note that these instances do not necessarily have to be hosts but could also be located inside the network providing for a less than end-to-end scope of the QoS channel. The arrows in Figure 1 symbolize signalling interactions between peering instances inside the network for a given level. The vertical interaction between levels as depicted by the dashed lines, i.e. the *concatenation* of signaling mechanisms, will usually be some kind of software mediating between the two adjacent signalling mechanisms, so upper- and lower-level signalling are co-located in such devices.

We only look at the constrained heterogeneity case where signalling protocols are layered on top of each other, since peer relations seem to have rather low relevance and are difficult to treat (for a discussion of this see [8]). Furthermore, they can usually be easily resolved by adding a minimal convergence layer to end-systems (as is e.g. the case for best-effort traffic if an ATM-attached host runs CLIP (Classical IP over ATM)).

2.2 General Issues

In a model of layered signalling as described above there are a number of issues that need to be taken into account.

An important issue is the *temporal characteristic* of the system, in particular how does latency incurred by an underlying signalling path involving non-local resources affect the realization of the overlaid signalling path. To obtain an understanding of the temporal characteristics of the overall layered system requires to understand the contribution of the different constituents of the overall signalling setup latencies and their quantitative relation to each other. If an underlying signalling system consumes a considerable amount of time relative to the time taken by the overlaid signalling system then one consequence is that *concurrent* handling of signalling requests at the interface to the underlying signalling system becomes a requirement. From the perspective of the overlaid signalling system there might be several sessions with outstanding admission control decisions. This introduces some complexities. One of them is with respect to strictly local resources like e.g. buffer space that may need to be set aside for a request. Such local resources should probably be acquired before the underlying signalling mechanisms are triggered in order not to waste presumably precious signalled resources if there are not enough local resources in the first place. A drawback of this design is however that a situation may occur where requests are denied service due to lack of local resources acquired by requests whose admission is still pending and might be rejected. Another implication of concurrently handling the admission control process of the overlaid signalling system is the necessity to introduce a further system state that might be called "*admission pending*". This state is required in order to avoid situations where sessions are torn down while the underlying signalling system is still busy establishing a lower level session. If the higher level session were torn down before the lower-level session signalling has been terminated then there would be no association any more for the lower-level session and it would be unclear how to deal with that session as no instance would feel responsible. Thus the admission pending state serves the purpose of not tearing down a session while a lower-level signalling system is still busy. One might argue that this case is extremely exceptional since a session setup and a session tear down will not likely follow each other so fast. However, if one takes into account that the overlaid signalling system might allow for dynamic QoS renegotiations or dynamic route changes as e.g. RSVP does, which may result in new session setups for the lower level signalling system, then the coincidence between higher level teardowns and lower level setups becomes much more frequent.

Another issue is the *coupling* between the higher and lower level signalling system. It need not be the case that every higher-level session is corresponding exactly to one lower level session and on the other hand a higher level session might consist of several lower level sessions. The first case where several higher level session are multiplexed onto a lower level session has the potential of lowering setup delays as at least for some sessions the admission control decision can be done based solely on local knowledge. On the other hand, it introduces a problem with QoS guarantees given to higher level sessions which are now multiplexed into one lower level session. This requires some mechanisms for careful grouping of higher level sessions in order to retain the individual QoS guarantees for higher level sessions (see [9] for a treatment of this). The second case where a higher level session consists of multiple lower level sessions may e.g. occur if a

heterogeneous multicast session needs to be overlaid on a lower level signalling system that only supports a homogeneous multicast model. In this situation the heterogeneous multicast session is emulated on the lower level by establishing multiple homogeneous multicast session - one for each QoS level. In both cases of logical decoupling between the higher and the lower level signalling system it is important to keep track of the associations. For example, a rollback operation might be necessary if some sessions of a collection of lower level sessions corresponding to a single higher level session could not be established, or if a lower level session is torn down due to a system failure and all higher level sessions using that lower level session need to be informed.

The idea or rationale behind the following investigations is that knowledge about the constituents of the overall setup latency in a hierarchically layered signalling environment has implications on what measures have to be taken when implementing such a system. Of course this is not solely dependent upon technology but also upon the topology of the layered network (meaning how much of the overall system does the underlying system make up respectively the overlaid system).

It is important to realize that these issues can only be treated if one has a clear understanding of the performance aspects of layered signalling.

3. Implementing RSVP/IntServ over ATM

As an example of a layered network QoS signalling configuration we present the implementation of the RSVP/IntServ architecture on top of a native ATM network.

3.1 Overall Architecture

The whole system of layered signalling systems in terms of the conceptual model is shown in Figure 2. While the end-systems communicate their QoS requirements via RSVP signalling to establish a unidirectional QoS channel, this RSVP signalling is mapped inside the network onto ATM UNI signalling at an edge device and then mapped by the access switches onto PNNI signalling for inter-switch communication. While the UNI to PNNI signalling mapping is quite straightforward as they were designed to interoperate, the mapping of RSVP onto ATM UNI needs more thought and is what we actually implemented at the edge device.

A high level architecture of such an RSVP/ATM edge device is given in Figure 3. Here the main building blocks of the edge device are shown. Of course an RSVP protocol engine is a required component to process RSVP messages between edge devices and other RSVP-aware nodes on the data path. The RSVP core identifies situations under which the lower level ATM UNI signalling may

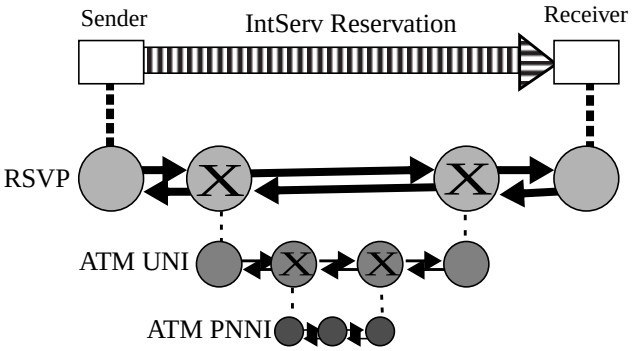


Figure 2: Layered signalling configuration for RSVP/IntServ over ATM.

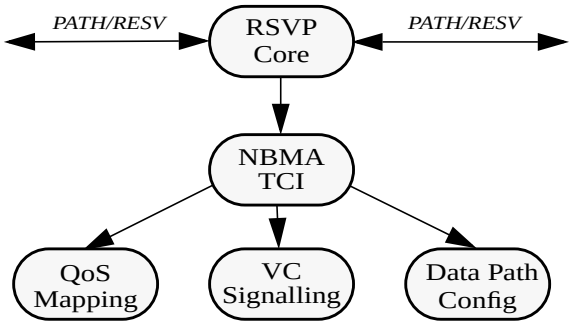


Figure 3: RSVP/ATM Edge Device Architecture.

need to be invoked and signals this to a component called NBMA TCI (Non-Broadcast Multiple Access Traffic Control Interface) which then makes the decisions with regard to whether an RSVP-related event actually requires an ATM signalling event and if so, what exactly should be done. The fact that almost any non-trivial subnetwork topology for an underlying QoS technology has the NBMA characteristic has been the reason for naming that component NBMA TCI. This component contains in its upper part generic functionality not yet specific to ATM, but is also valid for e.g. a DiffServ-capable subnetwork. However, the lower part of this module is ATM-specific and involves controlling subordinated modules for mapping the different QoS models, the invocation of ATM UNI signalling to establish VCs (Virtual Circuits) and to configure the data path inside the RSVP/ATM edge device.

In the following we first describe the implementation of a flexible IP/ATM adaptation which will cover the VC Signalling and the Data Path Config modules. We will shortly touch the QoS Mapping module as it is not the focus of this work and then present the design and implementation of the NBMA TCI.

3.2 The IP/ATM Adaptation Module

During the design and implementation of the IP/ATM adaptation module it became clear that it would not mean

much more effort to allow for a flexible software that could not only provide useful services to the RSVP/IntServ over ATM edge device but also to other IP-QoS related matters, as e.g. for DiffServ (Differentiated Services). So an effort was made to design the adaptation module as generic as possible. For a more detailed description of the capabilities and rationale of inner workings of the IP/ATM adaptation module see [10].

3.2.1 Implementation Design. We decompose the IP/ATM adaptation module, which we called *flexVCM* (flexible VC Management), into two separate parts:

- a *kernel module*, and
- a *user library.*

The *flexVCM* kernel module operates on the data path between the IP and ATM sides of an edge device and enforces the assignment of packets to VCs. A design decision we made was to implement the *flexVCM* kernel module in C. This was motivated by the fact that the kernel entry points are to be specified in C anyway and that it would make only limited sense to have a hybrid design by introducing another language, as e.g. C++.

The *flexVCM* user library acts on the control path by doing the signalling for VCs and furthermore provides the interface to users of the adaptation module. With respect to the programming language we decided to use C++, since we wanted an object-oriented interface design to be accompanied with object-oriented coding.

3.2.2 Interface. The *flexVCM* user library allows to set filters into the forwarding path from the IP-side of an edge device to the ATM-side. Here, filters consist of a number of rules which map data flows on a number of ATM VCs that can each be set up with a certain specified QoS. A user of the library only needs to supply the logic for which data flows there should be special treatment by the ATM subnet. This logic is a simple restricted predicate logic, where the predicates are based on arbitrary conditions in the headers including and above the IP layer and are combined by logical ANDs, thus constituting a *filter rule*. An OR'ed concatenation of such filter rules represents a *filter.* Each filter is mapped on a set of VCs, where the sets of the VC endpoints are disjoint. In a more formal way, filters can be described as:

Let $A_{i,j}(p)$, i=1,...,n, j=1,...,k, be predicates defined on the contents of an IP packet p,

$$\text{e.g. } A_{i,j}(p) = \begin{cases} 1 & \textit{if IP dest-addr = a.b.c.d} \\ 0 & \textit{otherwise} \end{cases}$$

then $F_j = A_1(p) \wedge \ldots \wedge A_n(p)$

constitutes a filter rule for j=1,...,k,

and $F = (F_1 \vee \ldots \vee F_k; VC_1, \ldots, VC_v)$

with $endpoints(VC_i) \cap endpoints(VC_j) = \{\}$ for all i,j

constitutes a filter.

Since flexibility is one of the design goals for the interface towards the *flexVCM*, different kinds of matching actual packet header's fields against filters are introduced, i.e. predicate definitions are very general. For example, it is possible to do mask matches which is particularly suited to address fields that are structured, as e.g. IP source and destination address fields, thus allowing for filter rules to be defined on whole IP subnets. Other types of matching include exact matches and range matches, where the latter could for example be used to specify a range of transport protocol ports.

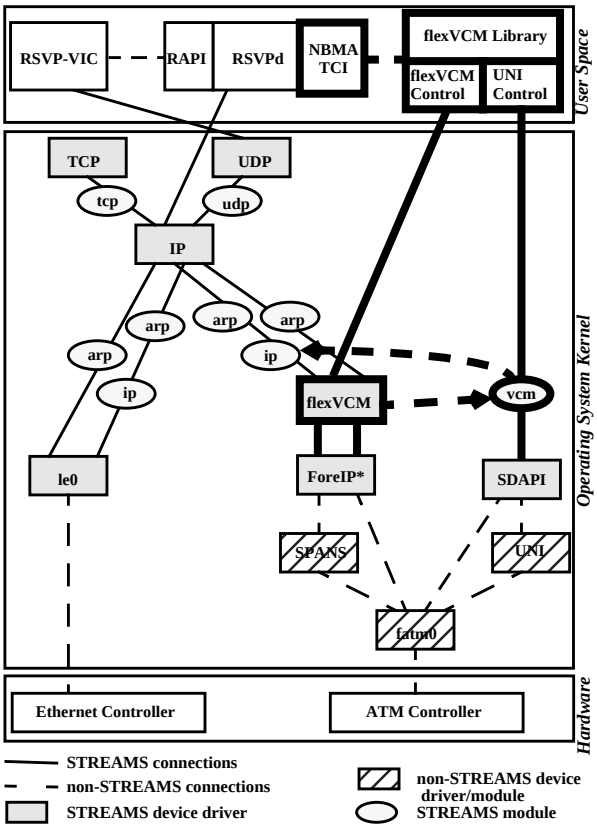
Let us briefly discuss why we chose to have a *N*:*M* relation between filter rules and VCs. The *N*, i.e. multiple rules, is due to the fact that it should be possible to share a VC by aggregating several flows onto the same VC(s), something which might be considered for RSVP/IntServ over ATM, especially for controlled load service (as proposed in [11]). The *M*, i.e. multiple VCs, which however do not share any endpoints, is because it should be possible to support a flexible way of combining IP with ATM multicast, as e.g. required if heterogeneous QoS multicast as provided by RSVP shall be supported efficiently by an ATM subnetwork [12]. The idea here is to construct heterogeneous QoS multicast trees from several homogeneous ATM point-to-multipoint VCs. Also note here that VCs might be shared between filters, i.e. a VC may belong to several filters. This means that e.g. for IP multicast groups that share only a subset of receivers it is still possible to share VCs to common subnet-receivers.

3.2.3 Implementation. Due to space restriction we are not able to present all the implementation details here. The interested reader is referred to [10] for a very detailed description.

Overall View. Our development environment is Sun work stations running Solaris 2.6/2.7 as the IP/ATM edge devices. The work stations are equipped with Fore's SBA200E respectively PCA200E ATM network interface cards. Therefore the *flexVCM* is realized as STREAMS implementation and for the VC control part we are able to use Fore's SDAPI (Signalling Driver API) as a means to interface directly to ATM UNI 3.1 signalling[1]. The components realizing the functionality of our IP/ATM adaptation module are depicted in Figure 4 with bold frames, whereas the other components represent the Solaris TCP/IP stack implementation and the ATM driver implementation by Fore.

The flexVCM Kernel Module(s). The most important component is the *flexVCM* STREAMS device multiplexing driver which is located between the IP multiplexer and a

[1.] UNI 4.0 is not available in the driver of our ATM adapters.



*:Here we could also use the classical IP or MPOA driver.

Figure 4: Implementation of *flexVCM*.

convergence IP module (in the example Fore IP is taken). The task of the *flexVCM* device is to multiplex the IP data streams according to configurable parameters onto ATM VCs. The IP multiplexer essentially does not see the Fore IP driver any more but is now communicating directly to the *flexVCM* multiplexer which however provides the same (DLPI) interface as the Fore IP device driver so that the IP multiplexer does not realize it "talks" to someone else. To take its multiplexing decision the *flexVCM* examines each IP datagram against a set of filters.

The flexVCM User Library. In Figure 5, the relevant part of the overall architecture for the *flexVCM* user library is shown. The *flexVCM* user library can on a macroscopic level be divided into two layers:

- The *user interface layer* which is directed towards a user of the *flexVCM* services as e.g. the RSVP daemon that "talks" via the NBMA TCI to the *flexVCM* module in order to set up special VCs for RSVP-signalled IP data flows according to the IntServ specifications carried by the RSVP messages. Another example (as depicted in Figure 5) could be a DiffServ SLA (Service Level Agreements) Manager that maps the given SLAs into specific ATM VCs.
- The *kernel interface layer* on the other hand is directed towards the kernel-level modules, respectively their

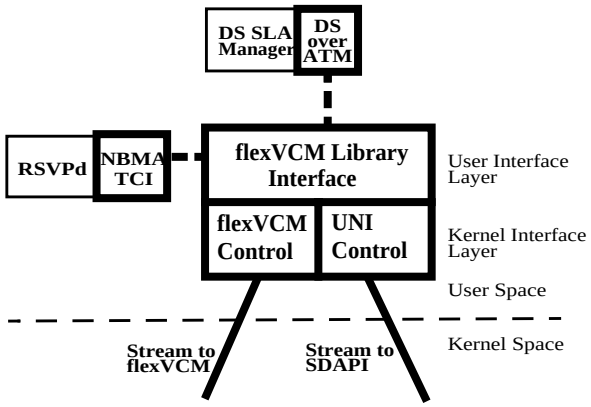


Figure 5: Global View of *flexVCM* User Library.

user-space interface, for managing the data forwarding inside the kernel and the setup and tear down, etc. of the especially customized VCs.

3.3 QoS Mapping

We used very simple, yet working mappings as described in [13], since the data path issues were not the focus of our investigations. Abstracting from details like AAL framing and segmentation overheads we used for Guaranteed Service CBR with $PCR = R$ and for Controlled Load CBR with $PCR = r$.[2]

3.4 The NBMA TCI

Several new aspects arise when broadening the point of view to traffic control for NBMA networks:

Silent Next Hops. Consider the arrival of the first RESV message from a downstream RSVP hop. Suppose that a reservation is already in place at the respective outgoing interface and that the new request carries no new FilterSpecs and a FlowSpec not larger than the existing one. For non-NBMA subnets at this point, no traffic control operation is necessary, because the new request can be served by the existing reservation. However, in case of NBMA networks, a new reservation request conveys a new next hop. This information must be handed over to the traffic control module, because it might be necessary to establish a dedicated transmission channel (i.e. a VC or VC-endpoint in case of ATM) to it.

IP Multicast. The interface to a traffic control module of RSVP is specified in [2]. With respect to IP multicast, it is mentioned in this document that the description “assumes that replication can occur only at the IP layer or ‘in the network’”. We denote this as a *broadcast* network. Without extensions, an RSVP engine merges all requests arriving at a single outgoing interface by calculating the least upper bound (LUB) of all FlowSpecs. In case of NBMA

[2.] Our ATM adapter cards did not support VBR VCs, which would be more efficient for both, GS and CL, than CBR VCs.

networks, though, the traffic control module itself must be able to decide how to merge reservations. We use the concept of *merging groups* to express this capability. Because ATM does not support point-to-multipoint-VCs with heterogeneous QoS parameters, the traffic control module partitions the set of next hops according to the similarity of their QoS requests into merging groups. Then, a point-to-multipoint-VC is used for transmission to each merging group.

Partial Rollback of Traffic Control. The scope of a single traffic control update operation is defined by the handling of a single RSVP message or timer expiration per interface. Performing such an update operation consists of several actions to be carried out. Besides installing a new FlowSpec for a reservation, FilterSpecs to identify eligible sender applications might be added or removed. Although unlikely, it is possible at least for a filter adding operation to fail. Therefore, we decided to prepare our traffic control modules for partial rollback by carrying out the update operation as follows:

First, the new reservation FlowSpec is installed, then filters are added or removed. If installing a new FilterSpec fails, all previously installed FilterSpecs from this update operation are removed again and the FlowSpec is set to its previous value. Thus, the important all-or-nothing property of a traffic control update operation is guaranteed by internal rollback.

Concurrent Execution. The traffic control operations might involve a certain overhead, so that it seems desirable to execute them concurrently to the core RSVP operations. This however, requires to specify most of RSVP's state information and operations such that concurrent execution is possible. This parallelization of the RSVP core processing has been done along the network interfaces. This means there is one thread per interface which does all kinds of RSVP processing for that interface. If a certain session is involved then this session's state is acquired via a mutex and hence no other RSVP-related processing can take place on this session. This is important for an NBMA TCI as it may trigger a non-trivial underlying signalling system as in our case ATM. By blocking on a session we essentially introduce the already mentioned admission pending state into the RSVP processing and thus avoid the problem when a session is removed before an associated traffic control operation is terminated.

3.4.1 Design and Implementation. In earlier work we have reimplemented the RSVP protocol, the so-called KOM RSVP engine [14]. We employed an object-oriented design. The implementation is done in C++. It is available for Linux, FreeBSD, Solaris [15] and shows pretty favorable performance characteristics [16].

Traffic Control. The design of the traffic control is shown in Figure 6. As can be seen in this diagram, a common base class *TrafficControl* exists, which provides the following interface to the core RSVP engine (arguments and return types are not shown):

```
class TrafficControl {
    virtual updateReservation() = 0;
    virtual redoLastReservation() = 0;
    updateFilters();
    addFilter();
    removeFilter();
    updateTC();
};
```

This base class completely implements the high-level handling of insertion and removal of FilterSpecs. The methods *updateReservation* and *redoLastReservation* are realized in derived classes and implement the logic for merging of multiple reservations. They are specialized on broadcast or NBMA respectively, depending on the actual type of subnet an interface is attached to. The class *Scheduler* acts as a base class for different flavors of scheduling packages and provides a common interface to them:

```
class Scheduler {
    addFlowspec();
    modFlowspec();
    delFlowspec();
    addFilter();
    delFilter();
};
```

The public methods of class *Scheduler* are eventually realized by calling internal virtual methods, which in turn are implemented in derived classes. Furthermore, this class provides some common mechanisms like logging of events and high-level admission control. Class *SchedulerNBMA* adds some methods to this interface, which are needed for NBMA subnets only.

```
class SchedulerNBMA : Scheduler {
    addDestination();
    delDestination();
};
```

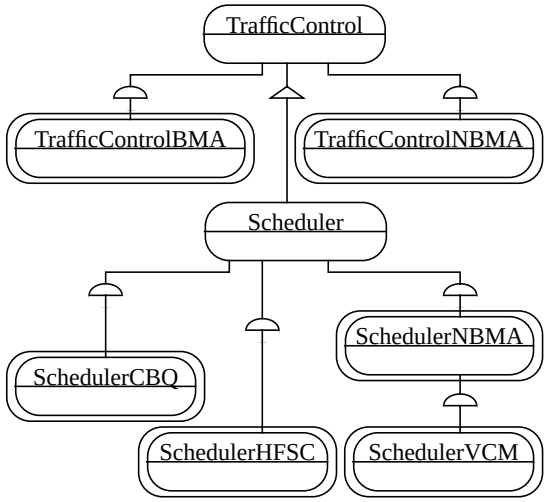


Figure 6: Class design for traffic control modules.

We have integrated scheduling packages for CBQ [17] on Solaris and CBQ and HFSC [18] on FreeBSD based on the ALTQ package [19]. However, most important to our discussion here is class *SchedulerVCM*, which represents the point where the contact from the RSVP daemon to the *flexVCM* module is established.

The integration between the NBMA TCI and the *flexVCM* module is in principle achieved by installing *flexVCM* filters in response to RESV messages arriving at the ingress RSVP/ATM edge device, where filters are composed of one or several filter rules (the latter is for shared explicit filter style reservation) defined by the 5-tuple (source IP address, destination IP address, source port, destination port, protocol). Furthermore, the *flexVCM* filters might in case of a heterogeneous multicast consist of several point-to-multipoint VCs, but at least two, one for best-effort receivers and one for QoS receivers. Due to the flexibility of *flexVCM*, all of this was easily possible. Furthermore, our RSVP over ATM implementation can interchangeably interwork with CLIP, ForeIP, or MPOA (Multiprotocol over ATM) for the transport of best-effort-traffic (in particular for the transport of RSVP control messages), as it is independent of the utilized best-effort IP convergence module.

4. Experiments & Measurements

In this chapter we conduct some experiments with a layered signalling system based on the implementation as described above. The focus will be on measurements of setup latencies and the analysis of contributions of different signalling steps to the overall setup latency. As a proof of concept and as a validation of the implementation, we also present some measurements on data path characteristics.

4.1 Setup Latencies

4.1.1 Test Scenario. Let us first describe the scenario we built up in the hope of mimicking a fairly realistic scenario with RSVP-capable access networks and an ATM backbone network. In Figure 7 the whole configuration is shown along with the flow of signalling messages on the different levels of the layered signalling system. The end-systems are Linux-PCs (Intel P6-III 450 MHz for the sender and AMD K6 350 MHz for the receiver) running a “vanilla” KOM RSVP daemon. They are connected via Ethernet to the RSVP/ATM edge devices which are Sun workstations (UltraSPARC 166 MHz for the ingress edge device and UltraSPARC-IIi 333 MHz for the egress edge device) running Solaris 2.6 and are equipped with the ATM-enhanced KOM RSVP daemon and an ATM adapter card as described in Section 3.2.3. The edge devices in turn are connected via OC3 interfaces to ATM access switches (the one closer to the sender is a Fore LE155 and the one closer to the receiver is a Fore ASX-200WG switch) and use UNI 3.1 signalling to communicate to the switches. The access switches are connected to each other via a backbone switch, a Fore ASX-1000, again using OC3 interfaces, and communicate to each other via PNNI.

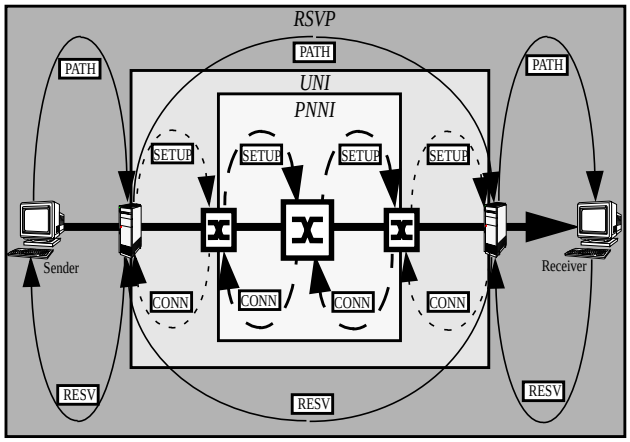


Figure 7: Test Configuration.

Let us now describe the sequence of events that take place in this system when a RSVP-based session is to be set up from sender to receiver:

1. A PATH message is generated from the RSVP daemon on behalf of the sending application and sent towards the receiver.

2. The PATH message is intercepted by the ingress edge device (ED), a PSB is created and the message is forwarded over the ATM subnetwork using a best-effort convergence module (we used CLIP).

3. The PATH message arrives at the egress ED, again a PSB is created and the message is sent on to the receiver.

4. The local RSVP daemon on the receiver obtains the PATH message, creates a PSB and upcalls the receiving application that is assumed to be already waiting for a PATH event.

5. The receiving application triggers immediately a reservation at its local RSVP daemon via the API.

6. The local RSVP daemon on the receiving machine, sets up an RSB and generates a RESV message which is sent towards the egress ED.

7. Upon reception of the RESV message the egress ED installs an RSB as well as an OutISB towards the receiver (using the CBQ-based traffic control module) and forwards the message to the ingress ED via the best-effort convergence module’s respective mechanisms.

8. The egress ED receives the RESV message, sets up an RSB, and now advises *flexVCM* via the NBMA TCI to set up a QoS VC with an appropriately translated FlowSpec.

9. A UNI SETUP message travels from the ingress ED to the access switch (a CALL PROCEEDING comes back).

10. The access switch now source-routes the SETUP mes-

sage towards the other access switch via the backbone switch using PNNI.

11. When the SETUP arrives at the egress ED this means that admission has been granted at all switches and the egress ED sends back a UNI CONNECT message.

12. The UNI CONNECT message travels back to the ingress ED (being transformed to a PNNI CONNECT and back on its way).

13. Upon arrival of the CONNECT message at the ingress ED, the VC has been setup completely and the FilterSpec and session descriptor parameters are used to instruct the *flexVCM* to redirect the RSVP-reserved flow over this new VC.

14. The NBMA TCI returns admission control success to the RSVP processing which then generates a RESV message towards the sender, which then upcalls the sending application with a RESV event, indicating that the reservation is now fully established.

4.1.2 Experiments, Results and Interpretation. For the test scenario as described above we have conducted different experiments which are designed to acquire an understanding of the different constituents of the overall setup latency. The experiments differ in respect to which components on the control path are actually turned on. Nevertheless the experiments are always conducted on the complete infrastructure of the test scenario thus avoiding differences due to different network characteristics and thus different transmissions delays.

In all experiments we sampled 100 measurements (in μsec) and computed the sample mean μ, the sample deviation σ, and a 95% confidence interval denoted as *CI*, using the student-t distribution since both mean and deviation are only sampled.

Experiment 1: RSVP on end-systems only. In the first experiment we only activated the RSVP daemons on the sending and receiving machine. The motivation was to see how RSVP performs in isolation. It was made sure that a CLIP best-effort VC had been set up already.

Table 1: Statistics for Exp. 1 Setup Latencies (in μsec)

μ	38538
σ	389
CI	(38313 , 38763)

The time taken was started when the sending application created the session until it received the RESV event (see Table 1).

Experiment 2: Vanilla RSVP on edge devices. In the second experiment standard RSVP processing was now activated at the RSVP/ATM edge devices, but the NBMA TCI was not activated and no special QoS VCs were built up. Again it was made sure that a CLIP VC existed between ingress and egress edge device.

Table 2: Statistics for Exp. 2 Setup Latencies (in μsec)

μ	47415
σ	2224
CI	(46126, 48704)

Interestingly, the increase of two more RSVP hops cost only about 9 ms (see Table 2), which despite the different machines being used shows that API processing seems to be quite expensive.

Experiment 3: RSVP-ATM adaptation at edge devices. In experiment 3 the RSVP-ATM adaptation was turned on at the edge devices so that now a QoS VC is set up in response to a RESV. This is the actual layered signalling system as depicted in Figure 7.

Table 3: Statistics for Exp. 3 Setup Latencies (in μsec)

μ	118943
σ	6896
CI	(114946, 122940)

The above numbers (see Table 3) indicate that the inclusion of the IP/ATM adaptation mechanisms has a strong effect on the overall setup latency. Two reasons are principally possible: the local RSVP/ATM edge device operations take a long time or the ATM signalling as triggered by the IP/ATM adaptation module takes a long time. Therefore another sample of more detailed measurements with regard to how much time is taken for the processing within the NBMA TCI as a whole and the contribution of the ATM VC setup was conducted (see Table 4).

Table 4: Statistics for Exp. 3 Setup Latencies with Different Contributions (in μsec)

Stat.	Setup Latency Over.	NBMA TCI	VC Setup
μ	117154	66589	66233
σ	8008	6344	6325
CI	(112513,121795)	(62912,70266)	(62567,69899)

Note that the NBMA TCI processing encompasses the VC setup in Table 4. It is obvious that the large increase in the overall setup latency is mainly due to the ATM signalling latency introduced by the three-hop ATM subnetwork. This is verified if we look at the difference statistics:

Table 5: Difference Statistics for Exp. 3 Setup Latencies (in μsec)

Statistics	Overall - NBMA TCI	NBMA TCI - VC Setup
μ	50564	356
σ	3843	41
CI	(48337, 52791)	(333, 379)

Here we can see that the NBMA TCI processing exclusive the VC setup is pretty inexpensive, while the VC

setup time is huge. These measurements certainly reinforce the issues we raised in Section 2 and tried to deal with in Section 3. Note that if we did not address the signalling latency of the underlying signalling system, here ATM, by parallelization of the RSVP engine (see Section 3.3.4) then the edge devices would only be capable of processing 15(!) setup/modification RESV messages per second. An alternative idea in order to circumvent the huge setup latencies might be to use "off the shelf" VCs that are pre-established and thus immediately available. Yet, this would require some knowledge of typical QoS requests. We did not investigate this any further.

Experiment 4: PNNI vs. no PNNI. In the fourth experiment we were interested in the contribution of PNNI signalling to the overall setup latency. We therefore configured a PVC between the access switches (yet still via the backbone switch). We measure again the overall setup latency and the different contributions of the whole NBMA TCI processing and its subcomponent, the VC setup.

Table 6: Statistics for Exp. 4 Setup Latencies (in μsec)

Statistics	Setup Latency	NBMA TCI	VC Setup
μ	76860	29679	29280
σ	3413	4077	4067
CI	(74882, 78838)	(27316,32042)	(26923,31637)

The results show that the overall setup latency is now considerably smaller, yet the contribution of the VC setup which now only involves UNI processing at the edge devices and the access switches is still high. We also give the difference statistics for the overall setup latency from exp. 3 and 4 in Table 7.

Table 7: Difference Statistics for Exp. 3&4 Setup Latencies (in μsec)

Statistics	Overall Exp. 3 - Overall Exp. 4
μ	40294
CI	(35262, 45326)

Of course the difference statistics here are from two independent samples, yet the student-t distribution still applies (though with one degree of freedom less) and we can compute the *CI* even if we cannot compute the sample deviation (as it makes no sense here). It can be noted that there seems to be no significant difference between UNI and PNNI processing at least in our configuration (~40 ms for two PNNI hops vs. ~30 ms for two UNI hops).

4.2 Data Path QoS

Although this is not the focus of our paper we want to present some experiments with respect to data path issues in order to complete the overall picture and to motivate the layered network configuration by the improved QoS provision on the data path.

4.2.1 Isolation of Flows. In this experiment a 5 Mbit/s constant rate UDP stream (1000 bytes/packet) was transmitted from sender to receiver for a 10 second duration. At the same time there was cross traffic from 2 sources (Sun Ultra workstations) directly connected to the ATM backbone switch. This cross-traffic flew to a traffic sink (again an Ultra workstation) connected to the access switch closer to the receiver. The cross-traffic from the two traffic sources was generated by independent on-off sources which both sent at a constant rate of 100 Mbit/s during their on-period of 2 seconds, while their off-period was 1 second. We measured the packet loss rate at the receiver for the case where a vanilla RSVP daemon was used at the RSVP/ATM edge vs. the case where the edge device implements layered signalling. The results in Table 8 are self-explanatory.

Table 8: Statistics of Loss Rate Behavior Experiment

Statistics	Packet Loss Rate vanilla RSVP	Packet Loss Rate layered signalling
μ	0.473	0
σ	0.041	0
CI	(0.449, 0.497)	(0, 0)

4.2.2 Packet Processing Overhead. The second point we looked at with respect to data path issues were the packet processing overheads incurred by the use of the *flexVCM* module. In particular we wanted to investigate the effect on best-effort traffic. Therefore we measured the time taken when a packet entered the *flexVCM* module until it leaves it again towards the best-effort convergence module. The results of 20 measurements are shown in Table 9.

Table 9: Statistics for packet processing times (in μsec)

Statistics	Packet Processing Time
μ	16
σ	0.3
CI	(15.7, 16.3)

The mean packet processing time was measured on the ingress edge device (equipped with 166 MHz UltraSPARC processor), thus corresponding to ~2700 cycles - a satisfactory performance. The same experiment but now for QoS-entitled packets yielded approximately the same results (see Table 10).

Table 10: Packet processing times of QoS-entitled packets (in μsec)

Statistics	Packet Processing Time
μ	17
σ	0.2
CI	(16.8, 17.2)

5. Related work

There is a a large set of related work concerned with interworking heterogeneous QoS architectures. Most of it is describing frameworks as e.g. in [20], some is reporting about implementation experience for RSVP/IntServ over ATM as e.g. [21], [22], or [23]. Yet, most of these were not implementing complete functionality, often only realizing unicast capabilities, and none of them provided any detailed performance measurements with respect to the control path. However as we argued in this paper such measurements are critical to the design and implementation of layered network QoS signalling systems.

Furthermore, there is to our knowledge no work that investigated the general issues of layering network QoS signalling protocols from a performance perspective, as we attempted here.

6. Conclusion & Future Work

In this paper we investigated the issues for layered network QoS signalling systems. We designed and implemented one particular instance of a layered network QoS system where the RSVP/IntServ architecture operates on top of an ATM subnetwork. In particular, we developed a flexible IP/ATM adaptation module and a general NBMA TCI as basic building blocks of an RSVP/ATM edge device mediating between the different QoS architectures. We then used this implementation to set up a test bed which allowed us to investigate the performance characteristics of the control path of such a layered configuration and to verify our design and implementation.

For future work we plan to develop a hopefully realistic performance model of layered network QoS signalling systems by implementing the interworking of RSVP/IntServ with a DiffServ/Bandwidth Broker based subnetwork which shall allow us to generalize our findings. Moreover, we will perform more detailed investigations on the data path characteristics of our implementation.

7. References

[1] D. Clark. The Design Philosophy of the DARPA Internet Protocols. *ACM Computer Communication Review*, 18(4), August 1988. Proceedings of SIGCOMM'88 Symposium.

[2] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource Reservation Protocol (RSVP) - Version 1 Functional Specification, September 1997. RFC 2205.

[3] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview, June 1994. RFC 1633.

[4] D. Black, S. Blake, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services, December 1998. RFC 2474.

[5] E. Rosen, A. Viswanathan, and R. Callon. A Proposed Architecture for MPLS, August 1999. Internet Draft, work in progress.

[6] Y. Bernet. The Complementary Roles of RSVP and Differentiated Services in the Full-Service QoS Network. *IEEE Communications Magazine*, 38(2):154–162, February 2000.

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

[8] J. Schmitt, L. C. Wolf, M. Karsten, and R. Steinmetz. A Taxonomy of Interaction Models for Internet and ATM Quality of Service Architectures. *Telecommunication Systems Journal*, 11(1-2):105–125, Jan 1999. Special Issue on European Activities in Interactive Distributed Multimedia Systems and Telecommunication Services.

[9] J. Schmitt, M. Karsten, L. Wolf, and R. Steinmetz. Aggregation of Guaranteed Service Flows. In *In Proceedings of the Seventh International Workshop on Quality of Service (IWQoS'99), London, UK*, pages 147–155. IEEE/IFIP, Jun 1999. ISBN 0-7803-5671-3.

[10] J. Schmitt. A Flexible, QoS-Aware IP/ATM Adaptation Module. Technical Report TR-KOM-1999-06, Darmstadt University of Technology, December 1999.

[11] L. Salgarelli, M. DeMarco, G. Meroni, and V. Trecordi. Efficient Transport of IP Flows Across ATM Networks. In *IEEE ATM '97 Workshop Proceedings*, May 1997.

[12] J. Schmitt, L. Wolf, M. Karsten, and R. Steinmetz. VC Management for Heterogeneous QoS Multicast Transmissions. In *Proceedings of the 7th International Conference on Telecommunications Systems, Analysis and Modelling, Nashville, Tennessee*, pages 105–125, Mar 1999.

[13] M. Garrett and M. Borden. Interoperation of Controlled Load and Guaranteed Service with ATM, August 1998. RFC 2381.

[14] M. Karsten. Design and Implementation of RSVP based on Object-Relationships. In *Proceedings of Networking 2000, Paris, France*. Springer LNCS 1815, May 2000. ISBN 3-540-67506-X.

[15] M. Karsten. KOM RSVP Engine: Performance Results, 2000. http://www.kom.e-technik.tu-darmstadt.de/rsvp/performance.html.

[16] M. Karsten. KOM RSVP Engine, 2000. http://www.kom.e-technik.tu-darmstadt.de/rsvp/.

[17] S. Floyd and V. Jacobson. Link-sharing and Resource Management Models for Packet Networks. *IEEE/ACM Transactions on Networking*, 3(4):365–368, August 1995.

[18] I. Stoica, H. Zhang, and T. S. E. Ng. Hierarchical Fair Service Curve Algorithm for Link-Sharing, Real-Time and Priority Service. *ACM Computer Communication Review*, 27(4), October 1997. Proceedings of SIGCOMM'97 Conference.

[19] K. Cho. A Framework for Alternate Queueing: Towards Traffic Management by PC-UNIX Based Routers. In *Proceedings of USENIX 1998 Annual Technical Conference, New Orleans, LA, USA*, June 1998.

[20] L. Berger, E. Crawley, S. Berson, F. Baker, M. Borden, and J. Krawczyk. A Framework for Integrated Services with RSVP over ATM, August 1998. RFC 2382.

[21] T.Braun and S.Giorcelli. Quality of Service Support for IP Flows over ATM. In *Proc. of the KIVS '97*, February 1997.

[22] J. Schmitt, M. Zink, L. Wolf, and R. Steinmetz. Quality of Service for Recording and Playback of MBone Sessions in Heterogeneous IP/ATM Networks. In *Proceedings of SPIE'S International Symposium on Broadband European Networks (SYBEN'98), Zürich, Switzerland*. SPIE, May 18–20 1998.

[23] H. Chow and A. Leon-Garcia. Integrated Services Internet with RSVP over ATM Shortcuts: Implementation Evaluation. *Computer Communications*, 22(9), June 1999.