Design and Implementation of a Flexible, QoS-Aware IP/ATM Adaptation Module

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

Abstract -- **The overlaying of IP-based networks onto ATM subnetworks is a network configuration pattern found increasingly often. While IP networks traditionally only offer plain "best-effort" service they are now evolving to offer more sophisticated services. Nevertheless, the exact mechanisms for providing QoS in IP networks are not yet settled and essentially non-existing in today's production-level networks, with the Internet being the most popular and important example. On the other hand, ATM networks have been designed from their inception to offer a wide range of QoS mechanisms. Thus, given the configuration of an IP overlay network over an ATM subnetwork, it is very attractive to leverage ATM's QoS mechanisms to alleviate IP's QoS problem, at least partially. The invocation of those mechanisms will be done on so-called IP/ATM edge devices which are exactly at the frontier between the IP and ATM network.**

In this paper we describe the design and implementation of a flexible, QoS-aware IP/ATM adaptation module. This adaptation module allows an IP/ATM edge device to forward IP datagrams depending on their (header) contents onto specifically set up VCs in a performant manner. To achieve performance, it is necessary to implement this module in kernel space, at least partially. On the other hand, it should be easy to use, for e.g., an RSVP/IntServ over ATM, or a DiffServ over ATM mapping module. Therefore, the adaptation module is split into two parts, a kernel-level part that handles all the time-critical tasks of data forwarding and a user-level part which gives access to the functionality provided by the adaptation module.

Keywords: IP/ATM edge device, QoS, RSVP/IntServ, DiffServ.

1 Introduction

Most IP-based production networks essentially still offer only best-effort service, and so does the largest IP-based network - the Internet. However, the Internet is becoming or even already is a commercially used ubiquitous communication infrastructure. A fact which will eventually require the Internet (or also large IP-based intranets) to be able to accurately predict its performance for business-critical applications, i.e., deliver stringent Quality of Service (QoS) guarantees for those applications.

On the other hand, the Asynchronous Transfer Mode (ATM) technology offers a rich set of QoS-enabling facilities. However, due to its homogeneity stipulation, it faces its degradation to a link layer which is being used by TCP/IP in the core of the internetwork where its accurate QoS mechanisms are needed most.

A possible result of that discussion may be:

IP lacks QoS, but has a wide distribution - ATM has QoS, but is not available end-to-end. Hence it seems very reasonable that IP takes ATM's assist in order to provide QoS, so that its huge user base can profit from ATM's facilities without the need of introducing ATM end-to-end.

This kind of interaction is often also termed *overlay model* with the important special case of mapping the RSVP/IntServ (Resource Reservation Protocol/Integrated Services) architecture onto ATM subnetworks [1].

In general, the problem of providing QoS in packet-switched networks can be separated into the distinct but related problems on the control and the data path. While the control path is very dependent on the utilized solution for QoS provision in the IP network, the data path issues are rather generic. Therefore, we developed in a first step an IP/ATM adaptation module that allows to instruct the forwarding path inside an IP/ATM edge device to direct IP data flows according to certain characteristics onto specifically set up ATM VCs.

2 Architecture and Design

2.1. Overall Design Goals

We can distinguish between problem-specific and general design goals for the IP/ATM adaptation module. Problem-specific goals are related to what we actually want to achieve with respect to the functionality of our IP/ATM adaptation module. General goals are related to desirable characteristics any software system is thriving to achieve, however we highlight those that are of particular importance for the adaptation module.

2.1.1 Problem-Specific Goals

The first and foremost design goal is certainly to offer a **rich functionality**, which is to have a means of using ATM's mechanisms and characteristics for any IP QoS related matters, examples of which could be:

- RSVP/IntServ (for an overview see [2]),
- ST-II+ [3],
- DiffServ(Differentiated Services) [4],
- Policy-based configurational/static QoS, e.g. using COPS (Common Open Policy Service) [5],
- Secure communications (e.g. for virtual private networks),
- Hybrid TCP/IP-ATM API, as, e.g. AREQUIPA (Application Requested IP over ATM) [6] or a similar scheme described in [7].

From the pretty diverse potential uses of the IP/ATM adaptation module it follows that **flexibility** should be one of the most important characteristics of its design. Flexibility here is meant with regard to:

- mapping of IP flows onto ATM VCs, i.e. many-to-many relationships between flows and VCs should be possible,
- description of what constitutes a flow, i.e., more or less arbitrary rules on IP and higher level headers should be possible to define a flow of data that shall be forwarded using one or more VCs.

Another more technically motivated design goal is to be **independent of the IP convergence modules** used for best-effort IP traffic delivery, i.e., the adaptation module should be capable of interworking interchangeably with any of the following:

- Classical IP over ATM (CLIP) [8],
- Multi-Protocol over ATM (MPOA) [9], or
- vendor-specific implementations, as e.g. ForeIP.

The idea behind the independence from the best-effort IP convergence module is to be able to make use of their different strengths when undertaking experimental research in the area of IP and ATM interworking. Of course, this independence also expands the general applicability of the adaptation module.

2.1.2 General Goals

Of course, the list of general design goals is virtually endless, however what we want to do here is to emphasize those that are of special significance to the development of the IP/ATM adaptation module. These are:

- **Reusability** of the code, since some parts could also be interesting to filtering software for firewalls or similar environments that need to deal with customizable forwarding decisions within an edge router.
- **Minimization** of **kernel**-level part, while **maximizing** the **user**-level part without sacrificing **efficiency** on the data forwarding path, i.e. only the most necessary changes to the forwarding behavior should be realized inside the kernel, while all the control functionality should be handled by the user-space part of the implementation. The rationale behind this goal is the ease of development and coding in user-space when compared to kernel space.
- **Extensibility** of the code is certainly a must, as for example the rules constituting a QoS-worthy flow will certainly experience changes and extensions.
- **Minimal invasiveness** with respect to existing kernel code. This is a pragmatic design goal which shall allows us to make the code available to the public domain.
- **Simple**, but **flexible interface** to the services provided by the IP/ATM adaptation module.

2.2. Overall Architecture

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.*

2.2.1 The flexVCM Kernel Module

The *flexVCM* kernel module operates on the data path between the IP and ATM sides of the edge device and enforces the assignment of packets to VCs. Apart from the design goals for the overall architecture which also apply to the *flexVCM* kernel module, there are also more specific design goals for the *flexVCM* kernel module:

- The **functionality** provided by the *flexVCM* kernel module should be kept **minimal**, but **complete** ("keep it lean and clean"). The goal was to design **atomic functions** which can be composed to an enhanced higher level service provided by the *flexVCM* user library. The rationale for this is the higher effort required for development and coding in kernel space when compared to user space implementations.
- Despite minimality, the *flexVCM* kernel module should offer as much **flexibility** as possible, especially with regard to the **specification** of **rules** that prescribe which packets belong to a flow for which special VCs are available (virtually any information contained in IP and upper layer headers should be possible to qualify for such special treatment by the ATM network). In particular, different kinds of granularity should be possible, e.g., traffic from certain subnets (identified by CIDR prefixes) should be a possible criterion as well as application subflows that are qualified by e.g. (transport protocol, source address, destination address, source port, destination port, and/or even RTP header fields).

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++.

2.2.2 The flexVCM User Library

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. Besides the overall design goals for the *flexVCM* module as a whole, we also have some more specific design goals for the *flexVCM* user library. Those are:

- The **interface** to the *flexVCM* user library should be **flexible** and **easy** to use. It should be **extensible** for user code since not all potential uses of the *flexVCM* module can be anticipated now. Therefore we decided to design it in an **object-oriented** fashion.
- The *flexVCM* user library should **hide** all the **details** of the *flexVCM* kernel module. Therefore one of the main tasks of the *flexVCM* user library is to **enforce** the **rules** implied by the overall design of distributing the functionality into user and kernel space and by the lean design of the *flexVCM* kernel module. Furthermore, the *flexVCM* user library **enriches** the services provided by the *flexVCM* kernel module.
- Another important goal when developing the *flexVCM* user library must be the **decent handling of failure** conditions, as, e.g. the case where a switch breaks down and all the VCs are torn down. The *flexVCM* user library must be able to signal these asynchronous events to a potential user of its services and must be able to indicate which VCs are actually affected.
- **Concurrency** of UNI **signalling** processing in order not to block a user of the library.

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. However, since the system-level interfaces to the *flexVCM* kernel module and to the UNI services are in C, the lower part of the *flexVCM* user library is rather procedural. Therefore, C++ which supports both paradigms of programming, procedural and object-oriented, was ideal for our case.

2.3. Interface to IP/ATM Adaptation Module

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 most important design goals for the interface towards the *flexVCM*, different kinds of matching actual packet header's partial 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's source and destination address fields, thus allowing for filter rules to be defined on whole IP subnets (e.g. "all traffic from subnet a.b.c shall take special VC v when being forwarded to subnet d.e.f"). 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.

Due to the generality of filter rules it is well possible that a certain IP packet matches several rules, in particular if the *flexVCM* is used by several instances, e.g. if DiffServ and RSVP/IntServ components are using *flexVCM* at an IP/ATM edge device at the same time. To resolve such conflicts we have introduced the concept of a cost for filter rules, which has to be set by the user of the *flexVCM,* as this represents a policy decision. The *flexVCM* ensures that always the least-cost filter is selected. A sample policy could e.g. be to assign RSVP/IntServ-related filters always a lower cost than DiffServ-related filters, thus enforcing that RSVP/IntServ-related special treatment of IP flows is always given priority over DiffServ-related QoS provision.

Let us now 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 will be particularly required for e.g. supporting DiffServ over ATM, but might also be considered for RSVP/IntServ over ATM, especially for controlled load service (as proposed in [10]). 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 [11]. 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 Implementation

Due to space restriction we are not able to present all the implementation details here. The interested reader is referred to [12] for a very detailed description.

3.1. 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 resp. 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 UNI 3.1 signalling. As ATM switches we used Fore’s LE155, as well as ASX-200 and ASX-1000 switches.

The components realizing the functionality of our IP/ATM adaptation module are depicted in **Figure 1** with bold frames, whereas the other components represent the Solaris TCP/IP stack implementation and the ATM driver implementation by Fore.

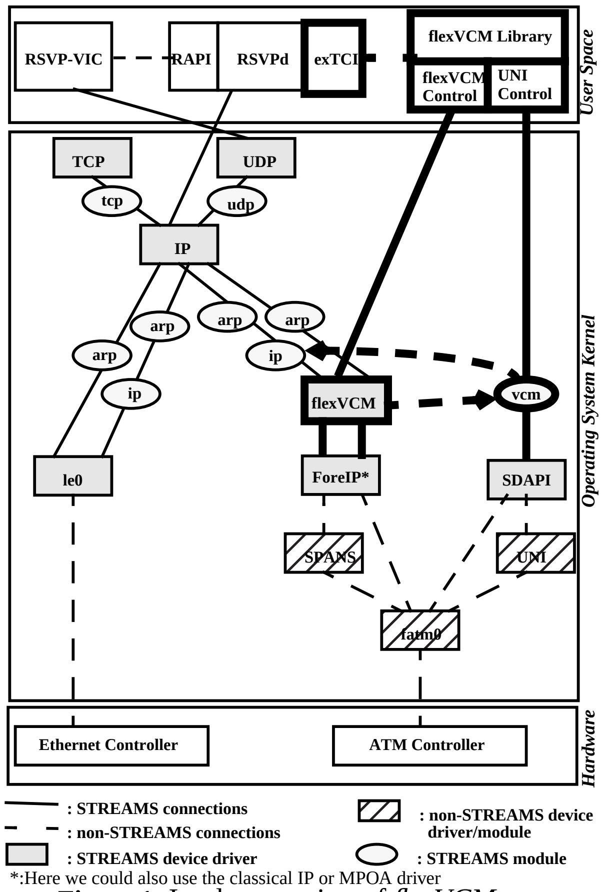


Figure 1: Implementation of *flexVCM*.

3.2. 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 convergence IP module (in the example Fore IP was 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.

The *flexVCM* multiplexer examines the IP datagram against a set of filters that are configured into it. The configuration of the filters is possible via an **`ioctl`** interface of the *flexVCM* multiplexer. If any of the filter rules applies, the *flexVCM* multiplexer directs the datagram onto the respective VCs of the least-cost filter rule which have been set up beforehand. If none of the filter rules apply then the datagram is just passed on to the best-effort IP convergence module driver.

For the redirection of the data over specifically set up VCs, the *flexVCM* multiplexer hands the successfully matched datagrams over to the *flexVCM* STREAMS module, which has been pushed on the SDAPI STREAMS device. In the *flexVCM* module the IP datagrams are prepared for being sent over their ATM VC by prepending an internal header required for the SDAPI driver. That is what has to be done for the ingress to the ATM network.

For the egress from the ATM network, the inverse has to be done by the *flexVCM* module: stripping off the internal header and putting the IP datagram into the upward directed stream to the IP multiplexer. These actions are depicted in **Figure 1** by the dotted arrows from the *flexVCM* device to the *flexVCM* module and from the *flexVCM* module to the data stream leading into the IP multiplexer.

The remaining question certainly is: who sets up the VCs and controls the filter configuration in the *flexVCM* device. This is done by the *flexVCM* user library. It uses the SDAPI provided by Fore to setup and manage VCs. These actions are recorded by the *flexVCM* STREAMS module and thus it is able to construct the required internal headers for use of the specifically set up VCs. The other task of controlling the *flexVCM* device by managing its filter set configurations is also done by the *flexVCM* user library.

3.3. The flexVCM User Library

3.3.1 Global View

In **Figure 2**, the relevant part of the overall architecture for the *flexVCM* user library is shown. The *flex-*

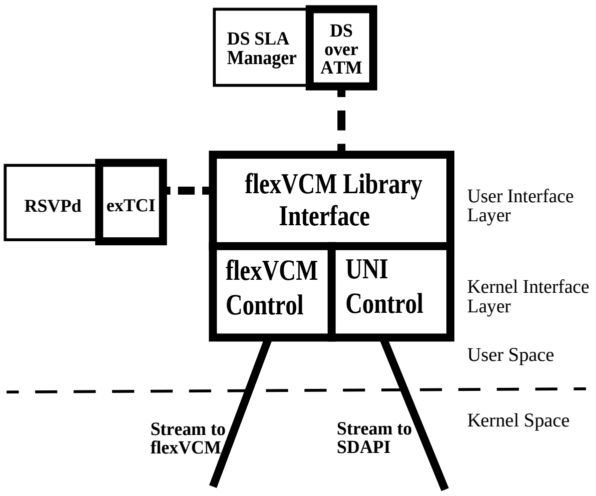


Figure 2: Global View of *flexVCM* User Library.

VCM 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 an extended Traffic Control Interface 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 2**) 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 user-space interface for managing the data forwarding inside the kernel and the setup and tear down, etc. of the especially customized VCs. Along those two different tasks it can be divided even further into subcomponents:
    - **`UNI Control`**, which handles everything that is concerned with the UNI-signalled VCs and which therefore uses the services provided by the user-level front-end to the SDAPI device (which itself communicates to the SDAPI device via the STREAMS concept), and
    - **`flexVCM Control`**, which handles all the necessary actions in order to invoke the *flexVCM* kernel module functionality of directing IP datagrams on specifically setup VCs depending on the (header) contents of these datagrams. Again the communication across the user-kernel space barrier is based on the STREAMS mechanism.

Since the SDAPI library provides an upcall-based interface that requires a user to "listen" on specific file descriptors periodically, it was decided to put this "polling" into an extra thread in order to not lock a user of the *flexVCM* while doing signalling for VCs in the ATM network. That means the **`UNI Control`** subcomponent of the kernel interface layer runs as a separate thread whereas all the other functionality runs as the main thread (separating the **`flexVCM Control`** subcomponent is possible but not necessary since the interface to the *flexVCM* kernel module works in a synchronous, immediate request-response fashion so that blocking for substantial periods is not an issue).

3.3.2 Static Model

A quite detailed static model, according to the Coad/Yourdon notation (see [13]), of the *flexVCM* user library is given in **Figure 3**.

The division between those classes that implement the kernel interface layer versus those classes that represent the user interface layer is illustrated by the bold dashed frame which encompasses all the classes that compose the kernel interface layer, while all the surrounding classes belong to the user interface layer.

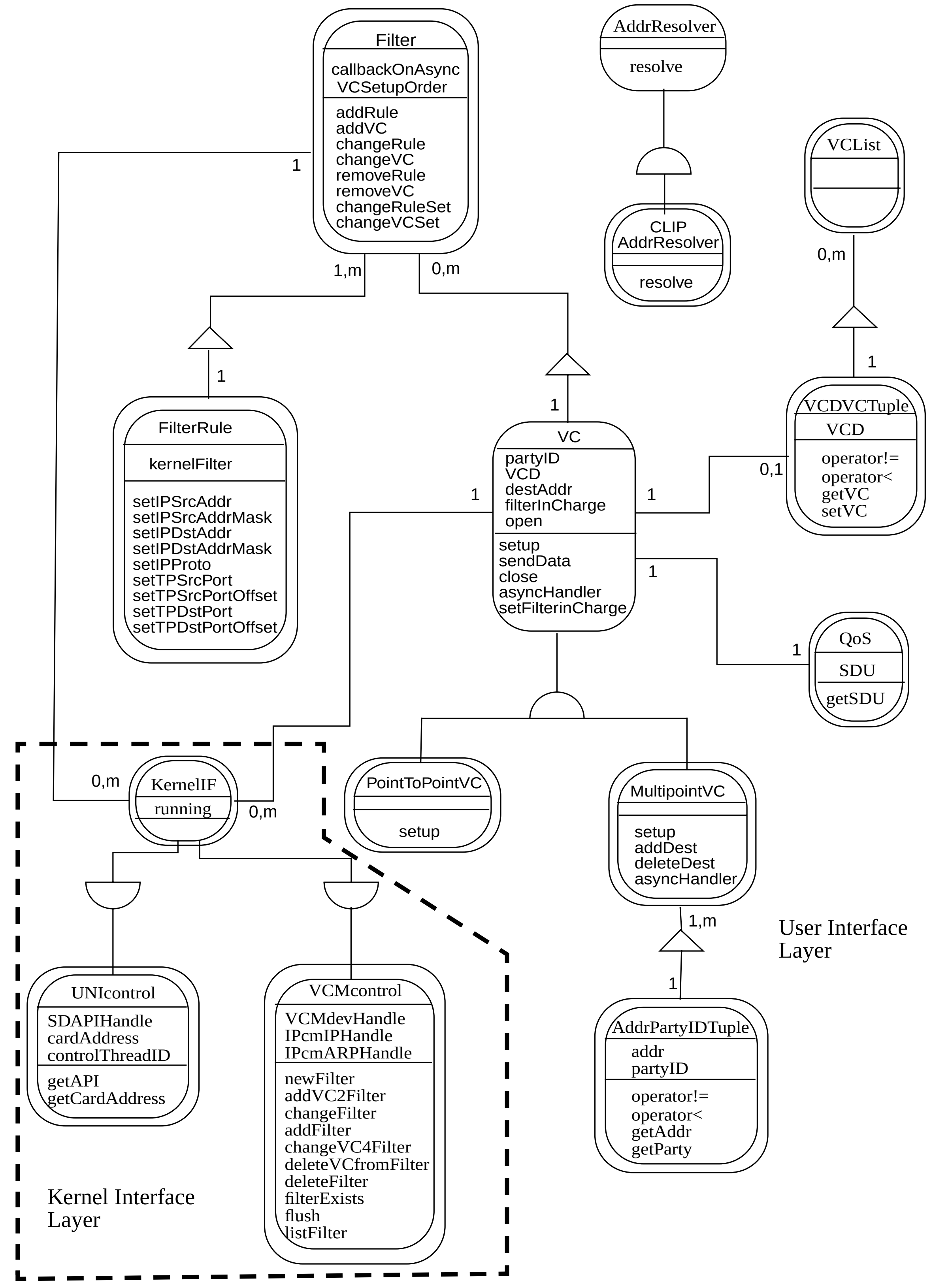
Let us briefly look at the relations between the most important classes, where by the term relation we mean here: inheritance, aggregation and object relations. The class **`KernelIF`** represents the kernel interface layer's interface. It inherits its functionality from the **`flexVCMcontrol`** and the **`UNIcontrol`** class which have a direct correspondence to the subcomponents forming the kernel interface layer's functionality as described above. Theses classes are implemented quite monolithic in a style that is not object-oriented in nature. That is due to their proximity to system-level interfaces, and actually the main task of these classes lies in building a convergence "layer" towards the purely object-oriented user-interface layer from the C-style system-level commands applying to the use of the SDAPI library and the *flexVCM* kernel module.

The user interface layer is composed of a larger variety of classes. The most important one being the **`Filter`** class. **`Filter`**'s consist of an arbitrary number (but at least one) of **`FilterRule`**'s and an arbitrary number of **`VC`**'s[1], thus storing the association between the rules on which IP datagrams are allowed to use which set of VCs. **`Filter`**'s have a relationship to the **`KernelIF`** in order to access its functionality for setting up kernel filters.

The **`VC`** class is an abstract base class for either **`PointToPointVC`** or **`MultipointVC`**. It has a one-to-one relation to the class **`QoS`** and a relation to the **`KernelIF`** in order to access its services for setting up VCs via UNI signalling. Furthermore, it has a relationship to the class **`VCDVCTuple`**, whose instances are kept in a container called **`VCList`** in order to track all the VCs irrespective of whether they were built up by the local

1. If the number of VCs is equal to 0, then the semantic of that filter is to discard a matching IP datagram, thus accomplishing the usual functionality of a packet filter for firewalling purposes.

Figure 3: The static model of the *flexVCM* user library.



flexVCM module or by another remote *flexVCM* module, i.e. for the former case they are outbound while for the latter case they are inbound. Only the outbound VCs are instances of the class **`VC`** which explains that a **`VCDVCTuple`** object does not necessarily have a relation to a **`VC`** object. The class **`MultipointVC`** which represents a specialization of **`VC`** contains an arbitrary number (but at least one) of **`AddrPartyIDTuple`** objects in order to track the association between ATM addresses and party IDs within a point-to-multipoint VC.

Isolated from the above classes of the user interface layer are the abstract base class **`AddrResolver`** and its specialization **`CLIPAddrResolver`**, whose task it is to resolve an IP address into the corresponding ATM address. The base class **`AddrResolver`** just specifies the abstract interface how such a request is to be made, whereas **`CLIPAddrResolver`** implements a simple way of eliciting the ATM address for a given IP address by using the CLIP's address resolution service. By separating the interface from the implementation, we are able to easily extend this to other address resolution mechanisms.

4 Example Use

We have used the *flexVCM* to implement two options for achieving differentiated treatment of IP flows over an ATM subnetwork:

- RSVP/Intserv, and
- static provisioning.

4.1. RSVP/IntServ over ATM

We have implemented an extended Traffic Control Interface (TCI) for NBMA networks like ATM which uses the *flexVCM* to grant reservations at an IP/ATM edge device a preferential treatment by the ATM subnetwork (for a detailed discussion of that TCI, see [14], which is based on [15]). This is achieved by installing *flexVCM* filters in response to RESV messages arriving at the ingress IP/ATM edge device, where the filter is composed of one or several filter rules (for shared explicit filter style reservation) defined by the 5-tuple (source IP address, destination IP address, source port, destination port, protocol). Due to the flexibility of *flexVCM*, this was easily possible. Furthermore, our RSVP over ATM implementation can interchangeably interwork with CLIP, ForeIP, or MPOA 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. As we are particularly interested in RSVP/IP multicast over ATM, we also implemented for CLIP a very simple way of supporting best-effort IP multicast. In any edge device a filter for all multicast traffic (224.*.*.*) is installed to use a point-to-multipoint VC to all other edge devices. This means of course, multicast traffic is essentially broadcast over the ATM subnetwork. While this is certainly not the most efficient way of supporting IP multicast over ATM, it is extremely easy to implement by the use of *flexVCM* (a few lines of code), which shows again the potential of the adaptation module.

4.2. Static Provisioning

The other example "application" of the *flexVCM* we implemented is an interactive console program for the *flexVCM*, which allows to install filters at an IP/ATM edge device exactly as they were described above. This can be considered as a way of statically provisioning resources inside the ATM subnetwork for certain IP flows by a network management operation. Due to the flexible way of specifying filters, this kind of provisioning can be done on almost arbitrary characteristics of IP flows, e.g. all IP telephony traffic could take an especially setup CBR VC using AAL1, whereas normal traffic would keep on using the configured IP convergence module and would therefore, e.g. be delivered over UBR VCs and AAL5. As it is also possible to use the ToS byte, or now the DS field, in order to discriminate between IP flows, the *flexVCM* console can also be viewed as a way of static configuration for an internal DiffServ node representing a particular ingress to an ATM subnetwork of a larger DiffServ domain.

5 Related Work

The issues surrounding the interworking between IP and ATM have been and are still dealt with extensively. However, directly related to our work, there are two main areas. One of them, is the general idea of so-called layer 4 or even layer 5 switching, as e.g. described in [16] or [17]. While this work is very interesting and partially complements our work with regard to the filtering algorithms described there, we are in contrast to these especially concerned with differentiated treatment of IP traffic entering an ATM subnetwork. The other area of related work is about interworking IP QoS architectures with ATM. With regard to approaches for RSVP/IntServ over ATM, see for example [18], [19], or [20], for DiffServ over ATM there is mostly conceptual work, as e.g. [21]. All of these focus very much on the control path issues and, if they provide implementation details at all, they solve the data path issues specifically for their regarded QoS architecture. In our paper, we provide an insight into the data path issues when implementing a flexible IP/ATM adaptation module that can in principle be used for any IP QoS related interworking with ATM.

6 Conclusion

We designed and implemented an IP/ATM adaptation module that is capable of supporting many situations in which IP QoS is to be overlaid onto ATM networks A special emphasis was put on the flexibility of the module. By implementing some example “applications” of that adaptation module, we showed that the design is flexible indeed and allows to be used for several IP QoS over ATM problems.

7 Outlook and Future work

The adaptation module is currently under performance testing and the feedback from this will certainly result in much fine-tuning. Our plan is to make the implemented software available to the public domain, which has been one of our design goals from the beginning and resulted in a design that tries to change as little existing kernel-level code as possible. Furthermore, with the help of *flexVCM* we want to experimentally research some hard control path problems, for which we devised conceptual solution approaches, e.g. in [22]. The latter was the original motivation to start work on the IP/ATM adaptation module in the first place.

8 References

[1] J. Schmitt, L. Wolf, R. Steinmetz, Y.-O. Lorcy, and C. Siebel. Interaction Approaches for Internet and ATM QoS Architectures. In *Proceedings of the 1st IEEE International Conference on ATM (ICATM’98), Colmar, France*. IEEE, June 22–24 1998.

[2] P. White and J. Crowcroft. Integrated Services in the Internet: State of the Art. *Proceedings of IEEE*, 85(12), December 1997.

[3] L. Delgrossi and L. Berger. Internet Stream Protocol Version 2 (ST2) Protocol Specification - Version ST2+, August 1995. RFC 1819.

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

[5] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry. The COPS (Common Open Policy Service) Protocol. RFC 2748, January 2000.

[6] W. Almesberger, J. LeBoudec, and P. Oechslin. Application REQUested IP over ATM (AREQUIPA), July 1997. RFC 2170.

[7] S. Martignoni and T.Kühnel. Extension of Classical IP over ATM to Support QoS at the Application Level. In *Proceedings of the 1st IEEE International Conference on ATM (ICATM’98), Colmar, France*. IEEE, June 22–24 1998.

[8] M. Laubach. Classical IP and ARP over ATM, January 1994. RFC 1577.

[9] ATM Forum Technical Committee: Multi-Protocol over ATM v1.0, July 1997.

[10] 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.

[11] 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*, March 1999.

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

[13] P. Coad and E. Yourdon. Object-Oriented Analysis, 1991. Prentice Hall, Englewood Cliffs.

[14] M. Karsten, J. Schmitt, and R. Steinmetz. Generalizing RSVP’s Traffic and Policy Control Interface. In *Proceedings of the 7th International Conference on Parallel and Distributed Systems (Workshops)*. IEEE, July 2000. Accepted for publication.

[15] M. Karsten. KOM-RSVP Protocol Engine, 1999. Work in Progress. Software available from http://www.kom.e-technik.tu-darmstadt.de/rsvp/.

[16] S. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast and Scalable Layer Four Switching. In *Proceedings of SIGCOMM’98*. ACM, September 1998.

[17] G. Apostolopoulos, V. Peris, P. Pradhan, and D. Saha. A Self Learning Layer-5-Switch. Technical Report RC 21461, IBM T.J Watson, April 1999. Research Report.

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

[19] 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.

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

[21] S. Ayandeh, A. Krishnamurthy, and A. Malis. Mapping to ATM Classes of Service for Differentiated Services Architecture, November 1999. Internet Draft, work in progress.

[22] J. Schmitt and J. Antich. Issues in Overlaying RSVP and IP Multicast on ATM Networks. Technical Report TR-KOM-1998-03, University of Technology Darmstadt, August 1998.