Generalizing RSVP's Traffic and Policy Control Interface

Martin Karsten[1], Jens Schmitt[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: {Martin.Karsten,Jens.Schmitt,Ralf.Steinmetz}@KOM.tu-darmstadt.de
http://www.kom.e-technik.tu-darmstadt.de/
http://www.ipsi.gmd.de/

Abstract -- **In this paper, we describe our current efforts to evaluate and extend the traffic control interface of RSVP and to define a useful policy control interface. The particular goal is to advance integration of non-broadcast multiple-access (NBMA) subnets, such as native ATM, and furthermore, to allow for meaningful inter-operation between traffic control and policy control. Our experience stems from implementing RSVP in combination with a policy interface to charging modules. Furthermore, we have developed a VC management module for ATM, which enables flexible inter-operation of RSVP with ATM subnets. We describe the current status of this work with respect to multicast, atomicity of operations and resulting challenges.**

I. Introduction

RSVP (Resource ReSerVation Setup Protocol), initially designed and described in [1], has been specified by the IETF [2] to carry reservation requests for communication resources across IP networks. The initial specification of RSVP lacks two aspects, which will be important for real operation of integrated services networks based on IP. The interface to traffic control modules, which eventually control and enforce resource reservations has been specified without taking into account the specific properties of non-broadcast multiple-access (NBMA) networks such as native ATM. In order to fully utilize the QoS capabilities of an ATM subnet, the original interface has to be extended. Additionally, no interface to policy control modules has been specified. A policy control module is needed to make and enforce administrative authorizations to use certain resources. As well, inter-operation with a charging system has to be carried out by this module.

Extending the traffic control interface as well as integrating an interface to policy control significantly increases complexity of operations at this point where an RSVP protocol engine interacts with external modules. For example, both traffic control and policy control have to admit a new flow entering the system. This decision must be done atomically, thus requiring a potential roll-back of one operation if the other module does not admit a new flow. Such advanced problems and the software design we have chosen to deal with them resemble the focus and main contribution of this paper.

While at first glance RSVP seems to be straightforward and easy to understand, the details of an implementation are rather complex. In order to cope with this complexity, we have developed a new design for an RSVP protocol engine, based on object-relationships [3], and we are implementing an RSVP protocol engine from scratch, employing this design. The latest release of this software package can be found at [4]. The next release will include the extensions described in this paper.

The rest of this paper is structured as follows. In the next section, we briefly review previous work related to practically dealing with traffic control and policy control interfaces. In Section III, we present those components that are integrated with our RSVP implementation. The concepts for integration are discussed in Section IV while in Section V, we present our software design and elaborate on certain aspects we learned from our implementation efforts, so far. Section VI concludes the paper with a summary and an outlook on further work items.

II. Related Work

Some work has been carried out to integrate native ATM subnets into RSVP operations [5,6]. However, these practical approaches do not address the problem in a general way to find efficient and flexible solutions. Our approach differs from earlier work in that we try to go beyond initial ad-hoc solutions and address the whole set of open issues, including for example, the impact on RSVP's message processing rules. We have described the initial design of an extended traffic control interface in [7] and have carried out a broader study of inter-operation issues in [8].

Design and operations of the RSVP implementation from ISI [9] are described in [10] and [11]. Further RSVP implementations exist [12], but other than the ISI project and ours, they are not in the public domain. In the area of policy control, the IETF working group RAP has published related work describing, among other things, a framework [13], design [14] and an RSVP interface [15] to policy control, but the specific issues discussed in the present paper are usually not addressed by these more generic documents.

III. Components for Traffic and Policy Control

In this section, we describe the additional components that are combined with the core of the RSVP protocol engine. First, we describe the main features of a generic IP/ATM adaptation module, then the charging system is briefly presented.

A. IP/ATM Adaptation Module

The interface to the IP/ATM adaptation module is implemented as a user level library that 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. The VC management (VCM) module takes all the necessary steps to set up corresponding VCs by using UNI signalling, rerouting the data path within the IP/ATM edge device, and so on. The logic is a simple restricted predicate logic, where the predicates are based on arbitrary conditions in the headers including and above the IP layer 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 disjunct. 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 the IP packet p,

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

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 the most important design goal for the interface towards the VCM, 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"). This general design allows the VCM to be employed for RSVP traffic as well as other QoS approaches for IP, such as Differentiated Services. A detailed description of this IP/ATM adaptation module can be found in [16]. At this point, we only briefly wrap up the most important VCM features:

- N:M relationship between filter rules and VCs for maximum flexibility of VC management strategies
- easy extensible and highly expressive filter rules by the use of predicate logic
- independence of general IP convergence module: Classical IP, ForeIP, Multi-Protocol over ATM
- ease of use: user-space object-oriented class library, yet performance-critical parts in kernel space

B. Charging System

A motivation and overview of charging for packet-switched networks can be found in [17]. We described an initial charging scheme for RSVP in [18]. Since then, the scheme has been extended to increase flexibility for additional pricing strategies (e.g. auctions) and to improve the inter-operation with a real RSVP implementation. Our approach mainly considers two aspects of the full challenge of creating a real-world charging system:

- price communication in the presence of RSVP's one-pass with advertising (OPWA) and receiver-initiated reservation strategy
- as precise as possible accounting of actual reserved resources to several adjacent hops with heterogeneous QoS requests in the presence of RSVP's reservation styles

The charging scheme consists of appropriate protocol elements and state information to describe prices, resource usage, etc. The policy control module offers several services to the RSVP engine. It allows for retrieving price information during the announcement phase of OPWA. During the reservation phase, the policy control module collects all merging information of reservations, extracts the respective resource accounting and calculates current pricing. Finally, the policy control module also decides from its point of view about authorization, acceptance and forwarding of a reservation request.

State information of our RSVP engine is modeled as objects and object-relationships, and hence, it is traversable. Therefore, it is sufficient to pass the same information to the policy control module as those passed to a traffic control module (roughly comparable to RSB and TCSB in [10]; see [3] for details). The operations of policy control can be separated into immediate and background tasks. This is further discussed in Section V.

IV. Conceptual Integration

Several new aspects arise when broadening the point of view to traffic control by NBMA networks and policy control. We describe the major challenges we encountered during our implementation project. In the rest of this paper, we use the terms *FlowSpec* and *FilterSpec* as defined in [2].

A. *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. If NBMA subnets and policy control modules are not considered 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 (e.g. a VC or VC-endpoint in case of ATM) to it. Also, a policy control module that calculates charges and accounts them to next hops must be informed about such a change, as well. Furthermore, state changes in the policy control module might affect the policy-related content of outgoing RESV messages.

B. *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. Note that a point-to-point link can be considered as special type of broadcast network, as well. As has been extensively discussed, e.g. in [8] and [19], there are many aspects of efficiently overlaying RSVP and ATM networks, which mainly result from the NBMA characteristics of ATM and the fact that ATM does not directly support the highly flexible IP/RSVP multicast model.

Without extensions, an RSVP protocol engine merges all requests arriving at a single outgoing interface by calculating the least upper bound (LUB) of all FlowSpecs. In case of NBMA networks, however, 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 e.g. ATM does not support Multicast-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 Multicast-VC is used for transmission to each merging group. Algorithmic aspects of efficiently building merging groups are studied in [20].

C. *Atomicity of Operation*

Both traffic control and policy control module independently decide about acceptance of a reservation, based on their respective state and configuration. Each operation must be done atomically, i.e., having an all-or-nothing property. Furthermore, for the core RSVP protocol engine, complete acceptance or rejection of a reservation must appear as a single decision, because RSVP has no mechanisms to deal with a reservation that is accepted by only one of both modules. Consequently, admission of a reservation request must be done in one atomic operation from RSVP's point of view. An open issue is to determine which of traffic control and policy control module decides first about admission and which is second. That module doing the first decision must be prepared for a full rollback if the other decision fails. We decided to place this burden on the policy control module for the following reasons.

It is likely to assume that policy control decisions generally consist of an authorization, a validity check and an accounting step. Validity check and accounting might be omitted, if the network operator considers it unnecessary. The validity check might be a test whether the offered payment is sufficient. This in turn requires part of the accounting process to be carried out. "Raw" resource accounting might be followed by an internal transaction, e.g., debiting an internal account. Internal transactions are periodically cleared by external transaction, i.e., real payments. Because an update of traffic control parameters immediately results in different network conditions, affecting other flows as well, it is favorable to lower the probability of a traffic control rollback over a policy control rollback. In case of a policy control rollback, internal transactions can be reverted without influencing any external entities.

Additionally, from the diversity of subnet technologies and their potential for complexity (as e.g. in ATM or when employing a subnet bandwidth manager) we conclude that the effort for rollback preparation and potential resource wastage depict an argument for this design decision. Yet another reason can be given by considering network provisioning. In a well-dimensioned network, traffic control rejections can be expected to be less likely than policy control refusals (e.g., because of overdue bills or empty pre-paid billing cards).

The same strategy has been chosen in the proposal for interaction between RSVP and COPS [15]. In the following we briefly present challenges resulting from those considerations.

1) 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 operations 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. Note that the above definition of a scope for a single operation prevents that filters are added and removed within the same update operation. As well, when filters are removed, the reservation FlowSpec is never increased.

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. By appropriately designing the respective software interface (see Section V), a traffic control module for NBMA networks can easily be integrated into this process to revert any merging group operations (as discussed in Section IV.B).

2) Full Rollback of Policy Control

In order to integrate a policy control modules that has the capability for rollback, the interface has to be split into two parts.

1. The preparation step consists of authentication and validity check. As a result, the request is either accepted or rejected and temporary state is saved within the policy control module.
2. The commit step corresponds to accounting, i.e., handing over the state information for persistent storing and potential external transactions.

If a reservation is accepted and later rejected by the traffic control module, it is sufficient to delete all temporary state information.

D. Concurrent Execution

Both traffic control and policy 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. To this end, we are actively investigating this topic, but have not come to any final solutions other than what is discussed in Section V.

V. Design and Implementation

The main design goals of our implementation are clarity of code, flexibility and extensibility. An RSVP implementation on a regular workstation using a normal UNIX operating system can only serve as a proof of concept and research platform for future investigations. Therefore, although we try to keep the design prepared for efficient operation, we do not believe that it is currently necessary to implement for outmost efficiency. We employ an object-oriented design and try to avoid any duplication of code. The implementation is done in C++.

A. RSVP Protocol Engine

State information of RSVP is stored as objects containing relationships to other objects. The contents of a PATH message are store in a Path State Block (PSB) whereas contents of a RESV message are stored in a *Reservation State Block* (RSB). As an example for relationships, each PSB has a relationship to a Previous Hop State Block (PHopSB) representing the hop from which this PATH message has been received. Information concerning a reservation at an outgoing interface is stored in an *Outgoing Interface State Block* (OutISB) and the relationship between reservations and PSBs is modeled as separate object *Outgoing Interface at PSB* (OIatPSB) in order to internally represent an N:M relationship by 1:N relationships (which simplifies implementation). Figure 1 shows the entity-relationship diagram for the design of RSVP state information.

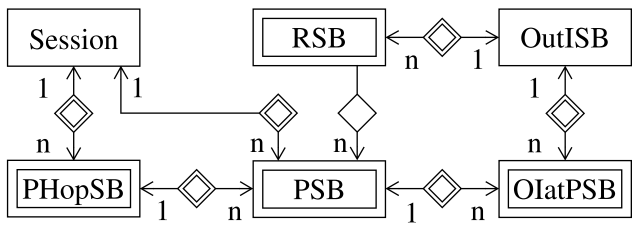

Figure 1: Entity-Relationship Diagram for State Blocks

B. Traffic Control

We model the traffic control and corresponding modules as a class hierarchy forming 4 layers of abstraction. Common tasks are implemented in higher layers whereas more specialized task are implemented in derived classes. This design is shown in Coad/Yourdon-Notation in Figure 2 without attributes and methods. As can be seen in this diagram, a common base class *TrafficControl* exists, which provides the following main interface to the core RSVP engine. We restrict the interface to the most relevant methods without showing details like arguments and return types.

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

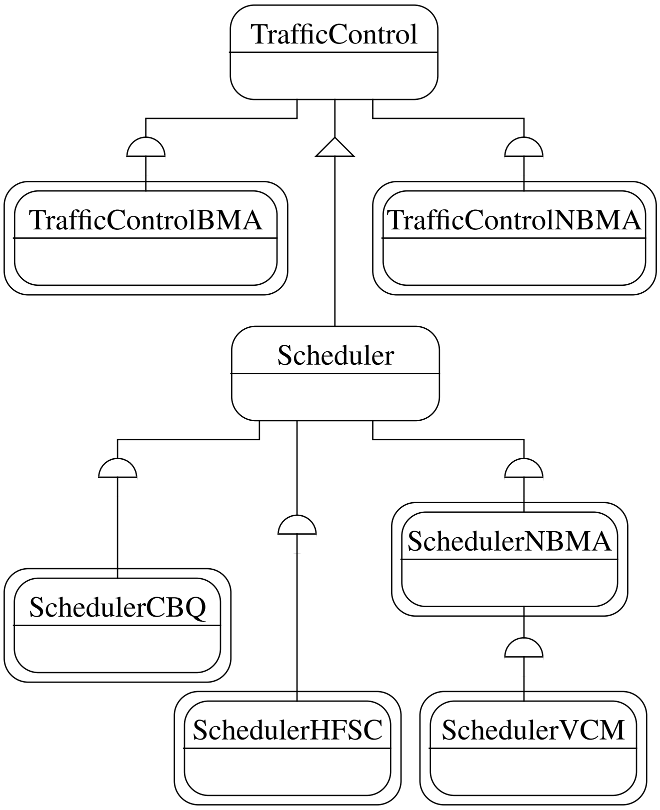

Figure 2: Class Design for Traffic Control Modules

```
   updateTC();
};
```

This base class completely implements the high-level handling of insertion and removal of FilterSpecs. Whenever during message processing a FilterSpec is found eligible for insertion and removal, a call to *addFilter* or *removeFilter* respectively is made. In order to minimize interaction between traffic control module and the system's resources, these actions are buffered within the *TrafficControl* class and executed only when *updateFilters* is called. 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. Correspondingly, two classes are derived from *OutISB*: *TCSB_BMA* and *TCSB_NBMA*. Internal state information for a reservation at an outgoing interface is stored in these classes. For example, merging group information for NBMA subnets is stored in objects of type *TCSB_NBMA*.

The class *Scheduler* acts as a base class for different flavors of scheduling packages and provides a common interface to them. This interface is basically the same as the traffic control interface in [10].

```
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();
};
```

To this end, we have integrated scheduling packages for CBQ scheduling on Solaris and FreeBSD and the VCM package on Solaris. We are currently in the process of integrating HFSC scheduling on FreeBSD.

C. *Policy Control*

The interface to policy control includes the necessary methods to perform policy control in two steps with a potential rollback after the first one. Given that our general design for the protocol engine allows to traverse object-relationships, it is suitable for those methods to take similar arguments as the traffic control interface. All operations that are carried out when *commit* is called, can be executed concurrently to further RSVP operations.

```
class PolicyControl {
   checkAndPrepare();
   commit();
   rollback();
};
```

D. *Structure of Operation*

In order to glue the pieces together, we present as pseudo code, how *updateTC* from class *TrafficControl* utilizes the services of other objects.

```
PolicyControl::checkAndPrepare();
if (success) {
   updateReservation();
   if (success) {
      updateFilters();
      if (success) {
         PolicyControl::commit();
         return;
      }
      redoLastReservation();
   }
   PolicyControl::rollback();
}
```

As can be seen from this pseudo-code, the appropriate design of traffic control and policy control modules and interfaces leads to a very concise and elegant expression of high-level concepts.

VI. Summary and Future Work

In this paper, we have presented some intermediate results from an ongoing evaluation and implementation project concerning policy control for RSVP and RSVP's operation over ATM subnets. We have described our implementation components, consisting of a new RSVP implementation, a flexible ATM/IP adaptation module, and a charging scheme that is currently being built.

We have identified challenges resulting from appropriate extensions of RSVP compared to its initial specification Then, we have discussed our approaches and solutions, as far as they are available. Finally, we have presented the software design for our implementation, which allows us and potentially others to further study those issues.

Future work items can be clearly identified. The theoretical work of finding appropriate merging groups over an ATM cloud should be backed up by simulation and/or real experiments. The charging system has to be completed to show its feasibility. The open issues about concurrency are yet to be investigated. Last not least, insights into overall efficiency and usability remain to be the primary goal in order to allow for real-world deployment of such mechanisms.

References

[1] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Network Magazine*, 7(5):8–18, September 1993.

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

[3] M. Karsten. Design and Implementation of RSVP based on Object-Relationships, 2000. Currently under submission.

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

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

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

[7] J. Schmitt and J. Antich. Extended Traffic Control Interface for RSVP. Technical Report TR-KOM-1998-04, Darmstadt University of Technology, July 1998.

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

[9] USC Information Sciences Institute. RSVP Software, 1999. http://www.isi.edu/div7/rsvp/release.html.

[10] R. Braden and L. Zhang. RFC 2209 - Resource ReSerVation Protocol (RSVP) – Version 1 Message Processing Rules. Informational RFC, September 1997.

[11] B. Lindell, R. Braden, and L. Zhang. Resource ReSerVation Protocol (RSVP) – Version 1 Message Processing Rules. Internet Draft, February 1999. Work in Progress.

[12] G. Gaines and M. Festa. RSVP-QoS Implementation Survey, July 1998. Available at http://www.iit.nrc.ca/IETF/RSVP_survey/.

[13] R. Yavatkar, D. Pendarakis, and R. Guerin. A Framework for Policy-based Admission Control. Internet Draft, April 1999. Work in Progress.

[14] J. Boyle, R. Cohen, D. Durham, S. Herzog, R. Rajan, and A. Sastry. The COPS (Common Open Policy Service) Protocol. Internet Draft, November 1999. Work in Progress.

[15] S. Herzog. RSVP Extensions for Policy Control. Internet Draft, April 1999. Work in Progress.

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

[17] M. Karsten, J. Schmitt, B. Stiller, and L. Wolf. Charging for packet-switched network communication - motivation and overview -. *Computer Communications*, 23(3):290–302, January 2000. to appear.

[18] M. Karsten, J. Schmitt, L. Wolf, and R. Steinmetz. An Embedded Charging Approach for RSVP. In *Proceedings of the Sixth International Workshop on Qality of Service (IWQoS'98), Napa, CA, USA*. IEEE/IFIP, May 1998.

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

[20] 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.