[SchA98] *Jens Schmitt:* **Aggregation of Guaranteed Service Flows;** Technical Report TR-KOM-1998-06, TU Darmstadt, November 1998.[SchA98]*Jens Schmitt, Javier Antich:* **Extended Traffic Control Interface for RSVP;** Technical Report TR-KOM-1998-04, TU Darmstadt, Juli 1998.

Darmstadt University of Technology

# Extended Traffic Control Interface for RSVP

*Jens Schmitt, Javier Antich*

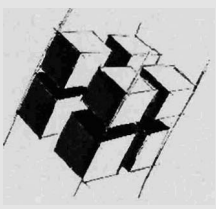
July 1998

Technical Report TR-KOM-1998-04

**Industrial Process and System Communications (KOM)**

Department of Electrical Engineering & Information Technology
Merckstraße 25 • D-64283 Darmstadt • Germany

Phone: +49 6151 166150
Fax: +49 6151 166152
Email: info@KOM.tu-darmstadt.de
URL: http://www.kom.e-technik.tu-darmstadt.de/



# Extended Traffic Control Interface for RSVP

*Jens Schmitt*[1]
1
Industrial Process and System Communications
Dept. of Electrical Engineering & Information Technology
Darmstadt University of Technology
Merckstr. 25 • D-64283 Darmstadt • Germany

{Jens.Schmitt}@kom.tu-darmstadt.de

**Abstract**

Internet and ATM both aim at providing integrated services. Therefore they independently (more or less) developed QoS architectures. A realistic assumption certainly is that both will take their place and that they will coexist for quite some time. A likely place for ATM is in the backbone, while IP will probably keep its dominance on the desktop. It is thus valid to assume an overlay model for the interaction between the two QoS architectures.

Crucial components of the QoS architecture of the Internet are its signalling protocol RSVP and the IP multicast architecture. There are several hard problems when trying to overlay this combination over an ATM subnetwork. In particular, the problem of matching RSVP's heterogeneous receiver concept onto the homogeneous point-to-multipoint VCs of ATM is such a problem. One solution to this problem is to provide VC management strategies to bridge that gap. However in order to be able to implement such VC management strategies RSVP's Traffic Control Interface and its message processing rules need to be extended to provide the necessary flexibility. These extensions will be presented in this report.

**Keywords:** QoS, Integrated Services, RSVP, IP Multicast, ATM.

## 1 Introduction

When considering the implementation of some of the VC management strategies introduced in in the companion report [Sch98] in support of heterogeneity over an ATM subnetwork, RSVP's Traffic Control Interface (TCI) and the relevant part of the protocol message processing rules as specified in ([BZB+97],[BZ97]) must be made more flexible than they are (this does not violate these standards, because these parts are only informational). Currently, RSVP merges all downstream requests and then hands the merged reservations to the traffic control module via the TCI. This leads to two problems if operating over ATM, or in general, a NBMA subnetwork with capabilities for multipoint communication:

- potential for not recognizing new receivers,
- solely support for the homogeneous QoS model.

These problems are already realized in [BZB+97], where it is conceded that the proposed TCI is only suitable if data replication takes place in the IP layer or the network (i.e. a broadcast network), but not in the link-layer as would be the case for ATM. Here, different downstream requests should not necessarily be merged before being passed to the traffic control procedures.

A new general interface is needed that supports both, broadcast networks and NBMA networks, where the replication can also take place in intermediate nodes (e.g. ATM switches) of the NBMA subnet. Only such modifications will allow for heterogeneity support over an ATM network, i.e. different VCs for different QoS receivers. However, even without taking into account heterogeneity support, there is a need for a modification of the TCI and the message processing rules due to the different nature of NBMA networks.

If a reservation request is received from a new next hop in the ATM network that is lower than an existing reservation for the session, then according to the currently proposed processing rules no actions

This work is supported in part by a grant of Volkswagen-Stiftung, D-30519 Hannover, Germany.

will be taken, since it is assumed that all the next hops within the same outgoing interface will receive the same data packets. That is of course not the case for an NBMA network like ATM, and some actions must be taken to add this new receiver to the existing point-to-multipoint VC. The same situation arises when a receiver tears down its reservation. If the LUB (least upper bound) of the other reservations does not change, nothing will be done with the current processing rules. However, the receiver must be deleted from the point-to-multipoint VC.

The problem with the current message processing rules and TCI is that, since they are based upon broadcast mediums, they do not allow any heterogeneity within a single flow and an outgoing interface. This is due to the fact that broadcast networks do not allow for heterogeneity of the transmission anyway. That is the reason why the LUB of the reservations requested for that interface is computed, thus making downstream merging.

A VC management strategy that supports heterogeneity does not need this downstream merging, or at least, no downstream merging of all the next hops in the interface. A more flexible scheme is necessary, that permits different "Merging Groups" within a certain interface. This general model includes the current model, if all next hops are considered as one merging group. A *Merging Group* (MG) is defined as the group of next hops with the same outgoing interface, whose reservation requests for a certain flow should be merged downstream, in order to establish a reservation.

For a single flow and outgoing interface, there may be several MGs. The two extreme cases are:

**a)** Only one MG: This is the case when no heterogeneity is allowed within the interface. Examples of this situation are:

- the homogeneous model when implementing RSVP over ATM,
- the underlying network technology is broadcast (e.g. Ethernet).

**b)** As many MGs as next hops: this would be the case if each of the next hops requires a dedicated reservation. Example applications of this are:

- NBMA networks which do not allow point-to-multipoint connections, and therefore, a point-to-point connection is needed for each of the receivers,
- the full heterogeneity model when implementing RSVP over ATM.

The most interesting options of this model from our point of view are the intermediate points between these two cases, where we allow a certain degree of downstream merging, so that it is possible to take advantage of the VC management strategies for heterogeneity support (Figure 1).

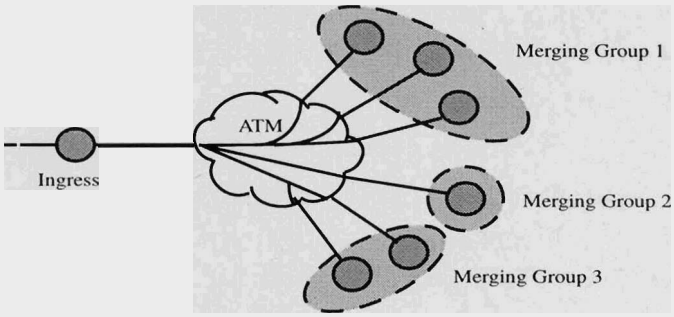



*Figure 1:* Merging Groups.

The TCI and the message processing rules should be independent of the number of MGs for a specific flow and the decision of including one next hop into a group or another should be taken by the traffic control module and not as part of the RSVP message processing. Details on how RSVP's TCI and its message processing rules need to be modified to allow for VC management strategies in support of heterogeneity will be discussed in section 2.

# 2 Extended TCI for Heterogeneous RSVP Flows over ATM Networks

This section will give the details on how RSVP's message processing rules and its Traffic Control Interface (TCI) need to be modified in order to allow for flexible VC management strategies for heterogeneity support.

## 2.1 RSVP's Traffic Control Interface

When analyzing how the combined architecture of RSVP/IntServ with IP Multicast and ATM can be integrated from an implementation's point of view, it is necessary to identify the parts of the RSVP specification that interact with the traffic control procedures offered by ATM. These are the RSVP message processing rules and RSVP's Traffic Control Interface (TCI) calls. RFC2209 [BZ97] describes the rules for the operation of Version 1 of RSVP (RFC2205 [BZB+97]). It outlines a set of algorithms which are induced by the rules of RFC 2205 and which should be used when implementing RSVP. RFC 2209 assumes the generic TCI calls defined in RFC2205 and some implementation-specific data structures. The description style in the following sections is aligned to that of the relevant RFCs to ease reading and comparison.

### 2.1.1 Traffic Control Interface Calls

RFC2205 presents a generic interface between RSVP and traffic control modules. Using these functions, RSVP can trigger the creation, change and deletion of reservations, as well as add or delete filters to existing reservations. The set of defined functions is:

```
TC_AddFlowspec( Interface, TC_Flowspec, TC_Tspec, TC_Adspec, Police_Flags )
-> RHandle [, Fwd_Flowspec]
```

- This function is used to establish a new reservation. Its main parameters are the **Interface** where the reservation must be set up, and the **TC_Flowspec** parameter, which specifies the desired effective QoS for admission control purposes; its value is computed as the maximum over the flowspecs of different next hops. The return value, **RHandle**, is an opaque number used by the caller for subsequent references to this reservation.

```
TC_ModFlowspec( Interface, RHandle, TC_Flowspec, TC_Tspec, TC_Adspec,
Police_flags ) [ -> Fwd_Flowspec ]
```

- This function is used to modify an existing reservation. The **TC_Flowspec** parameter is processed by the admission control procedure, and if the new reservation is rejected, the current flowspec is left in force. The corresponding filters, if there are any, are not affected by this function.

```
TC_DelFlowspec( Interface, RHandle )
```

- This call will delete an existing reservation, including the flowspec and all associated filter specs.

```
TC_AddFilter( Interface, RHandle, Session , FilterSpec ) -> FHandle
```

- Using this function, a new filter (source address and port) can be associated with the reservation corresponding to **RHandle**. The packet classifier module will use the existing filters of each reservation, to classify the packets into different flows, which will receive the appropriate QoS in the packet scheduler module. The return value, **FHandle** is a handle for subsequent calls to **TC_DelFilter()**.

```
TC_DelFilter( Interface, FHandle )
```

- This function would be called when a filter shall be removed from a reservation. The filter handle **FHandle**, returned from the **TC_AddFilter()** call, will be used for this purpose.

```
TC_Advertise (Interface, Adspec, Non_RSVP_Hop_flag ) -> New_Adspec
```

- This call is used for the OPWA (One Pass With Advertisement) mechanism to compute the outgoing advertisement **New_Adspec** for a specified interface.

`Upcall: TC_Preempt() -> RHandle, Reason_code`

- In order to grant a new reservation request, the admission and/or policy control modules may preempt one or more existing reservations. This will trigger a **TC_Preempt()** upcall to RSVP for each preempted reservation, passing the **RHandle** of the reservation and the subcode indicating the reason.

### 2.1.2 Data Structures

The data structures defined in RFC2209 which are significant to our investigation are:

`RSB - Reservation State Block`

- Each *RSB* holds a reservation request that arrived in a particular RESV message, corresponding to the triple:

`(session, next hop, Filter_spec_list)`

- Depending on the style of reservation, **Filter_spec_list** will contain:
  - **WF**: Nothing.
  - **FF**: Only one filter.
  - **SE**: A list of filters.
- The main contents of the *RSB* are:
  - session specification,
  - next hop IP address,
  - filter_spec_list,
  - outgoing (logical) interface OI, where the reservation is/has to be established,
  - style,
  - flowspec.

`TCSB - Traffic Control State Block`

- Each *TCSB* holds the reservation specification that has been handed to traffic control for a specific outgoing interface. In general, *TCSB* information is derived from *RSB*s for the same outgoing interface. Each *TCSB* defines a single reservation for a particular triple:

`(session, OI, Filter_spec_list)`

- The main fields of a *TCSB* are:
  - session,
  - OI - Outgoing Interface.,
  - filter_spec_list,
  - TC_Flowspec: LUB* over the flowspecs from matching *RSB*s,
  - RHandle, F_Handle_list.

Both, ***RSB*** **and** ***TCSB*** consist of additional fields described in RFC2209, but these are not important to the discussion of the next sections and were omitted for clarity. It should be noted that these data structures are implementation-specific and may contain different data members in particular implementations†. Other data structures like *PSB* (Path State Block) and *BSB* (Blockade State Block) are also described in RFC2209. For more details on these data structures and the ones explained above see section 1 of RFC2209.

---

*. LUB: Least Upper Bound of a set of flowspecs is the minimum flowspec that is larger than all the flowspecs of the set.

†. However most implementations are derived from ISI's code, which in turn accords to the above specifications, so that most implementations will "look" similar to this.

### 2.1.3 UPDATE TRAFFIC CONTROL Processing Rules

When a new reservation request arrives at a RSVP capable node, or a RESV-TEAR message is received, or a change occurs to any of the reservations established by this node, the last step before invoking the traffic control module through the TCI functions, is always the UPDATE TRAFFIC CONTROL sequence. The rules for this part of the RSVP processing are explained in RFC2209 section 2. In the following, a summary of this processing is presented, in order to simplify the understanding of the proposed modifications introduced later on. Some steps of this processing will be skipped in this summary. For more details see RFC2209.

The UPDATE TRAFFIC CONTROL sequence is invoked by many of the message arrival sequences to set or adjust the local traffic control state in accordance with the current reservation and path state. A parameter of this sequence is the **"active" RSB**.

If the result of the sequence is a modification of traffic control state, it notifies any matching local applications with a RESV_EVENT upcall. If the state change is such that it should trigger immediate RESV refresh messages, it also turns on the **Resv_Refresh_Needed** flag. These are the steps taken in the UPDATE TRAFFIC CONTROL sequence:

**a)** Compute the traffic control parameters using the following steps:

2. Consider the set of *RSB*s matching (session, OI) from the **"active" RSB**. The **Filter_spec_list** must also be matched if the style of the **"active" RSB** is FF. With these *RSB*s, compute:
   - the effective flowspec, **TC_Flowspec,** as the LUB of the flowspecs in the *RSB*s,
   - the effective traffic control *filter_spec list* **TC_Filter_Spec**, as the union of the **Filter_specs_lists** from these *RSB*s.

...

4. Locate the set of *PSB*s (senders) whose *SENDER_TEMPLATES* (i.e. address of senders) match **Filter_spec_list** in the **"active" RSB** and whose OutInterface_list includes OI.

...

6. Compute **Path_Te** as the sum of the *SENDER_TSPEC* objects (traffic parameters) in this set of *PSB*s.

...

**b)** Search for a *TCSB* matching (session, OI) and, if style is FF, also matches **Filter_spec_list**. If none is found, then create a new *TCSB*.

**c)** If the *TCSB* is NEW:

1. Store the values just computed: **TC_Flowspec**, **TC_Filter_spec**, **Path_Te** and other flags in this new *TCSB*.

2. Turn the **Resv_Refresh_Needed** flag on and issue the traffic control call:

```
TC_AddFlowspec ( OI, TC_Flowspec, Path_Te, police_flags)          ->
RHandle, Fwd_Flowspec
```

3. If the call fails, build and send a RESV-ERR message and delete the *TCSB*.

4. Otherwise, record the **RHandle** and **Fwd_Flowspec** in the *TCSB*. For each *filter_spec* in **TC_Filter_spec** call:

```
TC_AddFilter( OI, RHandle, Session, F ) -> FHandle
```

and record the returned **FHandle** in the *TCSB*.

**d)** If the *TCSB* is NOT NEW, but no *RSB*s where found in step a)2. , it means that the reservation

must be deleted:

1. Turn on the **Resv_Refresh_Needed** flag.
2. Call traffic control to delete the reservation:

```
TC_DelFlowspec (OI, RHandle)
```

3. Delete the *TCSB* and return.

**e)** The *TCSB* is NOT NEW, but the **TC_Flowspec**, **Path_Te** and/or police flags just computed differ from corresponding values in the *TCSB*, then:

1. If the **TC_Flowspec** and/or **Path_Te** values differ, turn on the **Resv_Refresh_Needed** flag.
2. Call the traffic control to modify the reservation:

```
TC_ModFlowspec ( OI, RHandle, TC_Flowspec, Path_Te,
                 police_flags ) -> Fwd_Flowspec
```

3. If the call fails, build and send a RESV-ERR message
4. Otherwise, update the *TCSB* with the new values and save **Fwd_Flowspec** in the *TCSB*.

**f)** If the *TCSB* is NOT NEW, but the **TC_Filter_Spec** just computed differs from the filter list in the *TCSB*, then:

1. Make an appropriate set of **TC_Delfilter()** and **TC_AddFilter()** calls to transform the **Filter_spec_list** in the *TCSB* into the new **TC_Filter_spec**
2. Turn on the **Resv_Refresh_Needed** flag.

...

**g)** ...

...

**h)** If the **Resv_Refresh_Needed** flag is on, the RESV REFRESH sequence will be invoked later on, and the appropriate RESV messages will be sent upstream.

## 2.2 Extensions to RSVP's TCI for NBMA Networks

### 2.2.1 The Traffic Control Interface and NBMA Networks

As explained in RFC 2205 sec.3.11.2, the details of establishing a reservation strongly depend upon the particular link layer technology in use on an interface. For multicast transmissions, there are three possible locations where data replication can take place:

**a) IP layer**: If packets are replicated at this level they will be sent onto different outgoing interfaces. The reservations coming from these interfaces must be merged to be forwarded upstream.

**b) Network**: Here replication takes place in the physical medium, e.g., an Ethernet LAN. In this case, the reservation requests within one outgoing interface (from different next hops) must be merged in order to establish the reservation for that outgoing interface and to forward the reservation upstream. Since the LUB reservation will be established on the outgoing interface some of the next hops will receive a better QoS than they requested.

**c) Link-layer driver**: This is the case of NBMA networks like ATM, where the data replication may occur in the link layer driver or interface card. Here, RSVP may need to apply different traffic control procedures for each VC independently, without merging requests from different next

RFC 2205 also points out that it would be desirable to organize an RSVP implementation into two parts: a core that performs link-layer-independent processing, and a link-layer-dependent adaptation layer.

The RSVP message processing rules as specified in RFC 2209, or more specifically their UPDATE TRAFFIC CONTROL part, are based on the TCI as specified in RFC 2205 sec.3.11.2, which explicitly assumes that the replication can only take place in the IP layer or the network. This means that not only the TCI, but also the message processing rules have to be modified in order to allow for a flexible implementation of RSVP over ATM.

A new general interface is needed that supports both, broadcast networks and NBMA networks, where the replication can also take place in intermediate nodes (e.g. ATM switches) of the NBMA subnet. Only these modifications will allow for heterogeneity support over the ATM network, i.e. different VCs for different QoS receivers. However, even without taking into account heterogeneity support, there is a need for a modification of the TCI and the message processing rules due to the different nature of NBMA networks. These basic changes will be explained first in the next sections before advancing to the broader modifications in order to allow for VC management strategies to support heterogeneity over the ATM network.

### 2.2.2 Changes in TCI and Processing Rules to Support NBMA Networks

In the current UPDATE TRAFFIC CONTROL sequence, after locating the different reservation requests(RSBs) for a specific session and outgoing interface (and a source template if the style is FF), the LUB of the different flowspecs of these RSBs is computed. Then, a TCSB corresponding to that session and OI is searched for. In the next steps, it is differentiated between three alternatives:

1. That no TCSB matching session and OI (and source for FF) is found. In this case, a new TCSB is created and the **TC_AddFlowspec()** function is called.
2. A matching TCSB is found, but there where no RSBs matching. Therefore, the previously computed LUB of the flowspecs is null and the list of filters for that reservation is also null. Under these circumstances, the **TC_DelFlowspec()** function is called.
3. A matching TCSB is found, and the new flowspec is different from the flowspec contained in the TCSB. This means that the reservation must be changed, and the **TC_ModFlowspec()** is called.

Now, with the ATM interface and taking into account multicast, a new case appears:

- A reservation request is received from a new next hop in the ATM network (see Figure 2). The LUB of the reservation requests coming from the ATM network is computed, and, let us suppose, it does not change. That means that the new request is lower or equal than the already existing reservation. With the currently proposed processing rules no actions will be taken, since they expect that all the next hops within the same outgoing interface will receive the same packets. That is of course not the case for an NBMA network like ATM, and some actions must be taken to add this new receiver to the existing point-to-multipoint VC. The same situation arises when a receiver tears down its reservation down. If the LUB of the other reservations does not change, nothing will be done with the current processing rules. However, the receiver must be deleted from the point-to-multipoint VC. Therefore, a new function

  `TC_Update_Destinations()`

  must be implemented, in order to add/delete nodes to/from the point-to-multipoint VC.

This little modification is sufficient for the support of a homogeneous QoS over the ATM network, i.e. if there is only one point-to-multipoint VC for a RSVP flow. However, for the support of multiple VCs per RSVP flow deeper modifications are necessary..

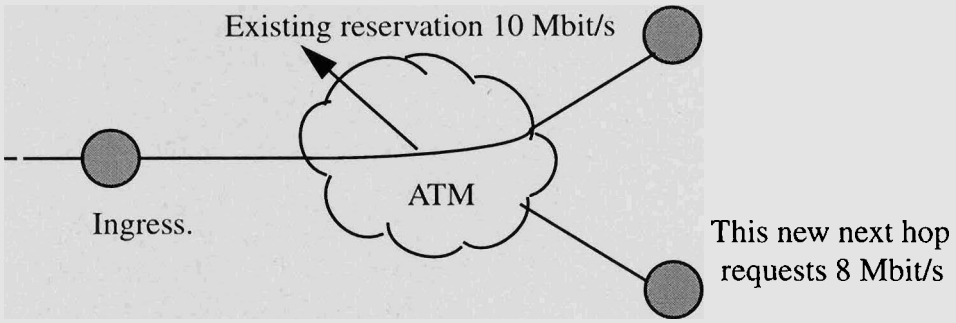



*Figure 2:* The problem of a new next hop.

### 2.3 Extensions to RSVP's TCI for Heterogeneity Support over NBMA Networks

The problem with the current message processing rules and TCI is that, since they are based upon broadcast mediums, they do not allow any heterogeneity within a single flow and an outgoing interface. This is due to the fact that broadcast networks do not allow for heterogeneity. That is the reason why the LUB of the reservations requested for that interface is computed, thus making downstream merging.

A VC management strategy that permits heterogeneity support does not need resp. cannot work with this downstream merging, or at least, no downstream merging of all the next hops in the interface. A more flexible scheme is necessary, that permits different "merging groups" within a certain interface. This general model includes the current model, if we use only one merging group. First of all, it is necessary to define what we mean exactly by the term "merging group":

> We define a **Merging group** (MG) as a group of next hops within an outgoing interface, whose reservation requests for a certain flow should be merged downstream, in order to establish a reservation.

For a single flow and outgoing interface, there may be several MGs. The two extreme cases are:

**a)** Only one MG: This is the case when no heterogeneity is allowed within the interface. Examples of this situation are:

- The homogeneous model when implementing RSVP over ATM.
- The underlying network technology is broadcast (e.g. Ethernet).

**b)** As many MGs as next hops: this would be the case when each of the next hops requires a dedicated reservation. For example:

- NBMA networks which do not allow point-to-multipoint connections, therefore, a point-to-point connection is needed for each of the receivers.

The most interesting options of this model from our point of view could be the intermediate points between these two cases, where we allow a certain degree of downstream merging, and at the same time it is possible to take the advantages of heterogeneity support. The TCI and the message processing rules

should be independent of the number of MGs for a specific flow and the decision of including one next hop into a group or another should be taken outside the UPDATE TRAFFIC CONTROL sequence.

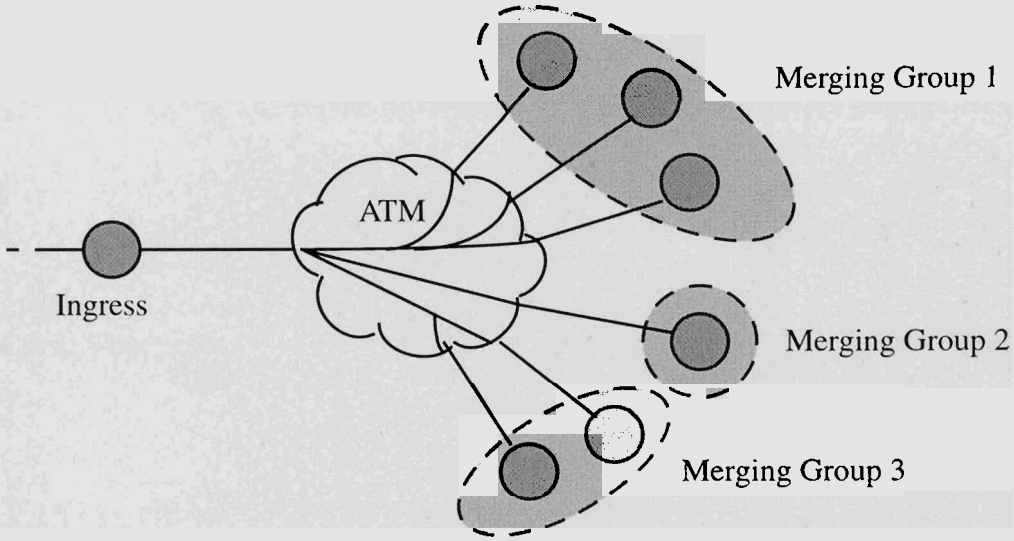



*Figure 3:* Merging Groups.

When a reservation change occurs and the UPDATE TRAFFIC CONTROL sequence is invoked, a new function should be called (**Update_MergingGroups()**) in order to determine in which MG this change took place. A change could be a new reservation request, a deleted reservation or a modified reservation. The **Update_MergingGroups()** function will be part of a particular implementation for the traffic control module, and should organize the different existing or new reservations into merging groups. There are a lot of possible ways of doing that, and it is a choice of the implementors, based on which network technology is available, and which degree of heterogeneity support, if possible, they desire.

Once the different reservations requested are distributed into MGs, the next steps are almost the same as the current processing rules. For each MG, its flow specifications should be merged and the union of the filters should be computed, in order to determine which of the functions for flow management (**AddFlow()**, **ModFlow()**, **DelFlow()**) and for filter management (**AddFilter()**, **DelFilter()**) should be called. Moreover, as already explained, a new function is needed, in order to update the destinations of the reservation within a MG, even if the effective flowspec of the group has **not** changed (**Update_Destinations()**).

Thus, the behavior of the UPDATE TRAFFIC CONTROL sequence in concert with the concept of MGs is independent of the particular link layer technology, which was the requirement. However, not only the new functions of the TCI (**Update_MergingGroups()** and **Update_Destinations()**), but also the already existing, are strongly dependent on the network technology. Therefore, different implementations are necessary for each kind of network (i.e. Broadcast, ATM, FrameRelay, ... ). Whether this distinction is made before or after the function call is an implementation detail.

Besides the changes in the processing rules and the TCI, the data structures utilized to maintain state of the reservations, also need some modifications. The Reservation State Block (*RSB*) and the Traffic Control State Block (*TCSB*) should include some information to distinguish among the different merging groups. The *RSB* could include an extra field, which identifies the MG within the interface to which the reservation belongs. The *TCSB* could be modified in two different ways. It could include a MG indicator, like an *RSB*, thus associating one *TCSB* with the reservation for a MG. Alternatively, we could keep the association between a *TCSB* and the pair (flow, outgoing interface), by modifying it internally to include the information of the different MGs in that interface, their filter lists and flow specifications. Even though both choices are principally possible, the first one is easier to implement.

Let us now investigate the modifications of the data structures, the message processing rules and the TCI in more detail.

### 2.3.1 Modified Data Structures

First of all, some extensions to the GENERIC DATA STRUCTURES as defined in Section 1 of [BZ97] have to be introduced:

- ***RSB - Reservation State Block*:** Two new fields, **MG_id** and **old_MG_id**, have to be included, as explained in 2.3, identifying the merging group to which the RSB belongs, and the MG this *RSB* has just left (see next sections for more details). Furthermore, one of the possible values of this MG identifier for reorganization is defined as NOT_ASSIGNED, since newly created *RSB*'s do not belong to any MG until the **TC_Update_MergingGroups()** function has been called.
- ***TCSB - Traffic Control State Block***: The same extension is required for the *TCSB*, in order to determine to which MG this traffic control state belongs, but in this case the **old_MG_id** field is not necessary.

### 2.3.2 Modified Traffic Control Interface

In this section, the necessary modifications and extensions to the TCI, in order to allow for a VC management strategy to support heterogeneity over a NBMA network, are introduced

```
TC_Update_MergingGroups( Session, *activeRSB )
```

The objective of this function is to carry out a MG management strategy, by e.g. including the new or modified *RSB* in the appropriate MG or creating a new MG. Existing MGs could even be changed depending on the VC management strategy in use. If a *RSB* is moved from a MG to another, the fields **MG_id** and **old_MG_id** must be filled correctly, so that the reservations can be correctly modified.

```
TC_AddFlowspec(Interface, MG_id, TC_Flowspec, TC_Tspec, TC_Adspec,
Police_Flags, activeRSB ) -> RHandle [, Fwd_Flowspec]
```

- This function interface is roughly the same as the one proposed in RFC2205, but the merging group identifier (**MG_id**) has been added, to provide the traffic control module with the information to which MG this new reservation belongs. Moreover, the **activeRSB** has been included in order to give access to the next hop information. For networks like e.g. Ethernet, the reservation does not require a connection to a specific endpoint, therefore the information of the next hop has no relevance. However, a more general interface should pass this information, in case the network is a NBMA, which necessitates the knowledge about the destination for a specific connection to be set up and thus to be able to establish the requested reservation.

```
TC_ModFlowspec(Interface, MG_id, RHandle, TC_Flowspec, TC_Tspec, TC_Adspec,
Police_flags ,activeRSB ) [ -> Fwd_Flowspec ]
```

- The parameters **MG_id** and the **activeRSB** have also been added to this function, for the same reasons as in the **TC_AddFlowspec()** function. The functionality is the same as explained in RFC2205. However, for NBMA networks, like ATM, this function also performs the set up and tear down of connections, depending on whether a new next hop has requested QoS or an old one has deleted its reservation. Or maybe even, because of changes in MG membership, after the **TC_Update_MergingGroups()** function has been called.

```
TC_DelFlowspec(Interface, MG_id, RHandle )
```

- In this function the parameter **MG_id** has also been added, for the reasons given above. This function is called when there are no more next hops in a MG, and therefore the reservation for that group can be deleted.

```
TC_AddFilter( Interface, MG_id , RHandle, Session , FilterSpec ) -> FHandle
```

- The parameter **MG_id** has again been included. Filters might be specific not to a flow, but to a MG. Due to the fact that each MG has a different set of receivers, for each MG, if SE style is used, the filters' union might be different. Thus, it is possible not to send a packet on a VC corresponding to a MG, if the members of that group have not included that source in their filters' list.

```
TC_DelFilter( Interface, MG_id, FHandle )
```

- The **MG_id** parameter has been included here for the same reasons as in the **TC_AddFilter()** function.

```
TC_Update_Destinations( Interface, MG_id,  Nhops_list )
```

- With this function the appropriate actions will be carried out (addition or deletion of nodes to/from a multipoint connection) to match the destinations of the **Nhops_list** and the nodes of the connection. This **Nhops_list** should be computed in the UPDATE TRAFFIC CONTROL module, the same way that the filters list is computed, i.e. per MG.

### 2.3.3 Modified Message Processing Rules

With the new TCI defined in the previous section, the RSVP Message Processing Rules (RFC 2209 [BZ97]) also require some changes. The purpose of these modifications is to provide a set of message processing rules, as general as possible, allowing for the support of NBMA networks and heterogeneity. Some difficulties appear if we do not know anything about what the **TC_Update_MergingGroups()** is doing, that is to say, to which extent the merging groups can be modified when this function is called. One could think of algorithms which dynamically reorganize the MGs depending on the requested resources, the current cost of the connections and the number of MGs.

In order to advance a first step with heterogeneity support, the modifications introduced in the message processing rules, assume that the **TC_Update_MergingGroups()** function will only modify the active RSB, either by assigning a new MG, or adding/deleting it to/from an existing group, or even changing from one MG to another. This limitation simplifies the changes in the message processing rules considerably.

The following shows how the UPDATE TRAFFIC CONTROL processing rules (section 2 of [BZ97]) look like:

1. For the active RSB call:

   ```
   TC_Update_MergingGroups( Session, active RSB )
   ```

2. Set **currentMG** = MG_id of the active RSB, and execute steps 8 to 15.
3. If step 2 failed

   3.1. Restore the MG_id in the active RSB with the old_MG_id, and return to the event sequence that invoked this one.

4. If step 2 did not fail,

   4.1. If old_MG_id and MG_id of the active RSB are different, and old_MG_id is not NOT_ASSIGNED , set **currentMG** = old_MG_id and execute again steps 8 to 15.

5. If the active RSB contains a RESV_CONFIRM object, then:

   5.1. If the Is_Biggest flag is on, move the RESV_CONFIRM object into the TCSB and turn on the Resv_Refresh_Needed flag. (This will later on cause the RESV REFRESH sequence to be invoked, which will either forward or return the RESV_CONFIRM object, deleting it from the TCSB in either case.)

   5.2. Otherwise, create and send a RACK message to the address in the RESV_CONFIRM

object. Include the RESV_CONFIRM object in the RACK message. The RACK message should also include an ERROR_SPEC object whose Error_Node parameter is the IP address of OI from the TCSB and that specifies "No Error".

6. If the Resv_Refresh_Needed flag is on and the RSB is not from the API, make a RESV_EVENT upcall to any matching application:

```
Call: <Upcall_Proc>( session-id, RESV_EVENT, style, Flowspec,
                     Filter_spec_list [ ,POLICY_DATA] )
```

where Flowspec and Filter_spec_list come from the TCSB and the style comes from the active RSB.

7. Return to the event sequence that invoked this one.

8. Compute the traffic control parameters using the following steps.

   8.1. Initially the local flag Is_Biggest is off.

   8.2. Consider the set of RSBs matching SESSION and OI from the active RSB. If the style of the active RSB is distinct, then the Filter_spec_list must also be matched.

   - If the active RSB has a FLOWSPEC larger than all the others, turn on the Is_Biggest flag.

   8.3. From this set of RSB's consider only those with **MG_id equals currentMG**.

   - Compute the effective traffic control flowspec, TC_Flowspec, as the LUB of the FLOWSPEC values in these RSBs.
   - Compute the effective traffic control filter spec (list) **TC_Filter_Spec*** as the union of the Filter_spec_lists from these RSBs.
   - Compute the **Nhops_list** as the union of the next hops of these RSBs.

   8.4. Scan all RSBs matching session and Filter_spec_list for all OI. Set TC_B_Police_flag on if TC_Flowspec is smaller than, or incomparable to, any FLOWSPEC in those RSBs.

   8.5. Locate the set of PSBs whose SENDER_TEMPLATEs match Filter_spec_list in the active RSB and whose OutInterface_list includes OI.

   8.6. Set TC_E_Police_flag on if any of these PSBs have their E_Police flag on. Set TC_M_Police_flag on if it is a shared style and there is more than one PSB in the set.

   8.7. Compute Path_Te as the sum of the SENDER_TSPEC objects in this set of PSBs.

9. Search for a TCSB matching SESSION, OI **and currentMG**, for a distinct style (FF), it must also match Filter_spec_list. If none is found, create a new TCSB.

10. If TCSB is new:

    10.1. Store TC_Flowspec, TC_Filter_Spec*, Path_Te, **currentMG**, and the police flags into the TCSB.

    10.2. Turn the Resv_Refresh_Needed flag on.

    10.3. Make the traffic control call:

```
TC_AddFlowspec( OI, currentMG, TC_Flowspec, Path_Te,
  police_flags, active RSB) -> Rhandle, Fwd_Flowspec
```

    10.4. If this call fails, build and send a RERR message specifying "Admission control failed" and with the InPlace flag off. Delete the TCSB, delete any RESV_CONFIRM object from the active RSB, and return.

10.5. Otherwise (call succeeded), record Rhandle and Fwd_Flowspec in the TCSB. For each filter_spec F in TC_Filter_Spec*, call:

```
TCAddFilter(OI, currentMG, Rhandle, Session, F)-> Fhandle
```

and record the returned Fhandle in the TCSB.

11. Otherwise, if TCSB is not new but no effective traffic control flowspec TC_Flowspec was computed in step 8, then:

11.1. Turn on the Resv_Refresh_Needed flag.

11.2. Call traffic control to delete the reservation:

```
TC_DelFlowspec( OI, currentMG, Rhandle )
```

11.3. Delete the TCSB and return.

12. Otherwise, if TCSB is not new but the TC_Flowspec, Path_Te, and/or police flags just computed differ from corresponding values in the TCSB, then:

12.1. If the TC_Flowspec and/or Path_Te values differ, turn the Resv_Refresh_Needed flag on.

12.2. Call traffic control to modify the reservation:

```
TC_ModFlowspec( OI, currentMG, Rhandle, TC_Flowspec,
  Path_Te, police_flags , active RSB) -> Fwd_Flowspec
```

12.3. If this call fails, build and send a RERR message specifying "Admission Control failed" and with the InPlace bit on. Delete any RESV_CONFIRM object from the active RSB and return.

12.4. Otherwise (the call succeeded), update the TCSB with the new values and save Fwd_Flowspec in the TCSB.

13. Otherwise,

13.1. Call:

```
TC_Update_Destinations(OI, currentMG, Nhops_list)
```

13.2. If this call fails, build and send a RERR message specifying "Admission control failed" and with the InPlace bit on. Delete any RESV_CONFIRM object from the active RSB and return.

14. If the TCSB is not new but the TC_Filter_Spec* just computed differs from the FILTER_SPEC* in the TCSB, then:

14.1. Make an appropriate set of TC_DelFilter and TC_AddFilter calls to transform the Filter_spec_list in the TCSB into the new TC_Filter_Spec*.

14.2. Turn on the Resv_Refresh_Needed flag.

15. Return.

As explained, these modified processing rules assume that only one RSB, the active RSB, is changed during the **TC_Update_MergingGroups()** function call. This requirement limits the algorithms that could be used within that function.

An algorithm which involves lots of changes in MGs' membership, would, as a result, also produce many modifications in the VC connections (new VC's, changes in point-to-multipoint VC, ... ). With such a scheme, it is essential to take care of what should be done in case of a failure of any of these changes, and how previous state can be restored. In order to solve the complexity introduced by this,

more comprehensive changes in the processing rules would be necessary. For example, the notion of a single active RSB is not useful any more. This concept refers to the RSB that had seen some kind of modifications (it was new, deleted or changed). However, with a complicated **TC_Update_MergingGroups()** an arbitrary number of RSBs can be modified, and all of them should be processed, the same way the single active RSB is currently processed.

All the difficulties that arise, when designing a TCI and processing rules valid for any model of heterogeneity support, may suggest that the UPDATE TRAFFIC CONTROL sequence might be different depending on the underlying network technology and the heterogeneity model utilized. Thus it would be more appropriate to include it into the traffic control module, thus integrating the downstream merging and reservation establishment tasks. With this scheme, the interface between RSVP and the traffic control module could be simply a single function **update_TC()** with the current parameters. This function would carry out a different processing for each traffic control module depending on the kind of network and/or heterogeneity support strategy.

# 3 Summary and Conclusion

This report is a very detailed description of how the RSVP Traffic Control Interface and the RSVP message passing rules need to be modified or rather extended in order to provide the flexibility that would be necessary to support VC management strategies in support of heterogeneity over the ATM subnetwork as described in [Sch98]. In this companion report we differentiated these strategies according to the fact whether the edge device is situated on the premises of the ATM network provider or not. That led us to different algorithms for each case. We showed how these algorithms could achieve a significant gain in either reduced costs or saved bandwidth when compared to simple schemes as proposed in the literature. That was the starting point for investigating the necessary changes in the RSVP over ATM implementation.

# References

[BZ97] R. Braden and L. Zhang. RSVP Version 1 Message Processing Rules, September 1997. RFC 2209.

[BZB+97] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource Reservation Protocol (RSVP) - Version 1 Functional Specification, September 1997. RFC 2205.

[Sch98] J. Schmitt and Javier Antich. Issues in Overlaying RSVP and IP Multicast on ATM Networks. Technical Report TR-KOM-1998-03, University of Technology Darmstadt, August 1998.