Experimental Extensions to RSVP — Remote Client and One-Pass Signalling
Martin Karsten
to appear in Proceedings of IWQoS'01, Springer LNCS

http://www.springer.de/comp/lncs/index.html

Experimental Extensions to RSVP — Remote Client and One-Pass Signalling

Martin Karsten

Industrial Process and System Communications, Darmstadt University of Technology
Merckstr. 25 • D-64283 Darmstadt • Germany
Martin.Karsten@KOM.tu-darmstadt.de

Abstract. We present and evaluate two experimental extensions to RSVP in terms of protocol specification and implementation. These extensions are targeted at apparent shortcomings of RSVP to carry out lightweight signalling for end systems. Instead of specifying new protocols, our approach in principle aims at developing an integrated protocol suite, initially in the framework set by RSVP. This work is based on our experience on implementing and evaluating the basic RSVP specification. The extensions will be incorporated in the next public release of our open source software.

1 Introduction

There have been numerous proposals for QoS signalling protocols, which exhibit differences along certain, partially interdependent, characteristics, for example with respect to participating entities, interaction mode, flexibility, generality, supported services, among others. The *Resource ReSerVation Protocol* (RSVP) [1] provides a rich set of functionality and has been chosen for standardization by the IETF [2]. However, the basic specification of RSVP considerably has shortcomings in a variety of contexts, in which the specific set of RSVP's features are either over- or under-dimensioned.

- Embedded end systems often have strict limitations with regard to their processing power and memory equipment. Therefore, it is imperative to keep the respective requirements of any signalling protocol as low as possible. Running a full RSVP daemon on such an end system might not be the appropriate configuration.
- A number of valid service models exist, in which the performance can be described as transmission rate over certain time intervals. In this case, RSVP's ability to collect path characteristics might not be needed. Furthermore, RSVP is designed to support multi-point to multi-point communication. This design requirement imposes a receiver-oriented reservation model and thus, a two-way session setup, which might not be needed for simple unicast communication. Therefore, a sender-oriented one-way reservation setup can be a sensible extension to RSVP.

The eventual goal of our work is to design an integrated protocol suite, which can be broken down to a few well-defined subsets for specific scenarios. Our current work is based on RSVP, because it seems to be a good candidate to start this investigation. We expect to either be able to actually design such a protocol suite within the framework set by RSVP, or alternatively, to gain important insight to design such a protocol suite from scratch, if RSVP turns out not to be an appropriate basis.

The rest of this paper is structured as follows. In Section 2, we present two extensions to RSVP. A performance-related evaluation is presented in Section 3 and the paper is wrapped up with a conclusion and an outlook to future work in Section 4.

2 RSVP Extensions

As discussed in Section 1, there are several circumstances under which the current RSVP specification is improvable to accommodate specific requirements. In this section, we present according protocol extensions for RSVP.

2.1 Remote Clients

RSVP defines two alternative methods to transmit messages between RSVP-capable nodes. RSVP messages are either transmitted as raw IP packets or using UDP encapsulation [2]. When using UDP encapsulation, packets are addressed to well-defined ports. If multiple clients run on a single end system, this addressing scheme requires a central manager entity (usually the RSVP daemon) to receive and dispatch incoming messages. For outgoing messages, it seems to be possible to use the same port numbers by multiple application processes, but this might not be supported on all platforms. For embedded devices, the effort of running a dedicated RSVP daemon might be prohibitively expensive, even if this daemon does not need the full functionality. An elegant solution is to define additional protocol mechanisms which allow an RSVP daemon running on the first RSVP-capable hop to administer and communicate remotely with a number of clients. These clients in turn only need to implement RSVP stubs and except for the special addressing scheme, participate in the full RSVP signalling procedure. This interaction is shown as *remote API* in Figure 1.

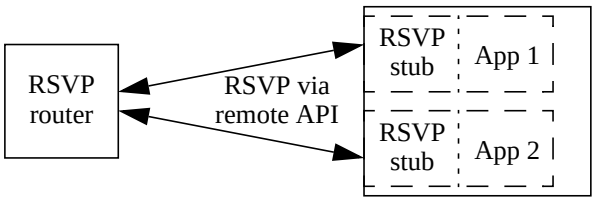

Fig. 1. Remote Client Extension

The remote client extension can be realized through a new message type, *InitAPI*, and reusing the LIH field of the RSVP_HOP object. In the notation of [2], the InitAPI message is defined as follows.

```
<InitAPI Message> ::=   <Common Header> [ <INTEGRITY> ]
                        <SESSION> <RSVP_HOP>
```

An additional flag in the SESSION object distinguishes whether a message is used to register or de-register a client. Of course, the detailed representation of protocol elements could be chosen differently, if necessary for any purpose. Both registration and de-registration messages carry the local IP address of the client system as part of the RSVP_HOP object. The LIH field of this object is used to carry the local UDP port, which is chosen arbitrarily by the clients. Clients communicate to the remote RSVP daemon through a well-known port. In general, from the point of view of the RSVP daemon, a client operates similar to a regular RSVP hop, distinguished only by the registration process and UDP communication. Client registration is done using soft state, i.e.

clients have to regularly refresh their registration, otherwise all respective state is timed out at the RSVP daemon. The periodic refresh is triggered by the RSVP daemon and other protocol messages are not refreshed between the daemon and the client in order to avoid complicated timer management at the client side. The application using the client API can optionally initiate retransmission of requests, if desired. In order to enable end-to-end consensus about established reservations, confirmation messages do not terminate at the daemon as in [2], but are forwarded to the client. Of course, the first-hop RSVP node must be in the path between the client and the other end system. Additionally, the client system is responsible for exerting traffic control on incoming reservation requests and allocating resources. This is identical to regular RSVP processing and even mostly independent of the signalling protocol at all, but rather on the actual link technology and its dimensioning.

2.2 One-Pass Reservations

In its basic form, RSVP uses a bidirectional message exchange to set up an end-to-end simplex reservation. This procedure is called *one-pass with advertising* (OPWA) [2] and used for the following purposes. In order to support heterogeneous requests from multiple receivers within a multicast group, reservations are requested and established from the receiver to the sender. The advertising phase is needed to route reservation requests along the reverse data path to the sender. Furthermore, to flexibly support a variety of service classes and to enable precise calculation of reservation parameters for delay-bounded services, appropriate data are collected during the advertisement phase and delivered to the receiver.

As discussed in Section 1, there are a number of scenarios in which both features are not needed. In such cases, the original OPWA procedure represents an unnecessary signalling overhead for both end systems and intermediate nodes. Additionally, there might be situations where an initial (potentially duplex) reservation establishment by the initiator is desirable as fast as possible, which can later optionally be overridden by appropriate signalling requests from the responder and in turn the initiator. We have designed a true one-pass service establishment mechanism, which allows to handle such situations. It fully interacts with traditional RSVP signalling, such that it is possible to optionally override an initial one-pass reservation with later requests. The operation of a one-pass reservation as duplex request is shown in Figure 2. The figure shows the situation for a responder overriding a reservation installed by the initiator. Below, we specify the protocol elements for this extension.

A new message type, *PathResv*, is defined to indicate that reservations based on the transmitted *TSpec* shall be established through the transmission of this message. Other than the message type, the syntax is exactly the same as for a *Path* message. In order to request a duplex reservation, the following object can optionally be added to a PathResv message

```
DUPLEX_Object ::=  <SenderReceivePort> <ReceiverSendPort>
```

The DUPLEX object carries the reverse port information, assuming that the same transport protocol is used in both directions. Again, this specification can easily be changed or extended, if necessary for any purpose. The duplex extension is only sensible, when symmetric paths can be assumed between two end systems and furthermore,

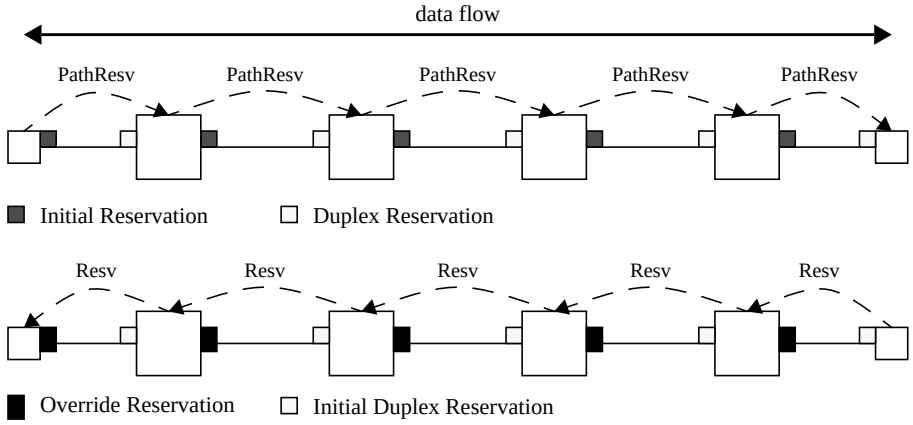

Fig. 2. One-Pass Duplex Request with Subsequent Override

only for unicast communication. Consequently, duplex requests for multicast sessions must be ignored at intermediate nodes.

The advantages of such an extension are quite obvious. First, it reduces signalling complexity for end systems, by offering a one-pass request model without active involvement of the responder. Optionally, a confirmation message could be send back to the initiator, in order to assure the end-to-end service establishment, but we have not implemented that, yet. By reducing the overall signalling effort to a single pass, intermediate nodes are relieved from processing effort, as well, because of fewer total messages. Thereby, this mechanism enables lightweight signalling in the framework of RSVP. These advantages are increased even further when one-pass duplex signalling is employed. Optionally, one-pass session establishment can be overridden by later requests from both initiator and responder. In this case, any state that has been indirectly created through one-pass mechanisms is replaced by regular state. While this usage scenario eventually leads to the same overall signalling costs as using traditional RSVP, it allows for a faster initial session establishment, because only one half of the round-trip is needed. As a side effect, the remote API extension also allows to better integrate legacy and new RSVP-incapable end-systems, because no interaction with low-level system services is needed to port it to such platforms.

3 Evaluation

The extensions presented in Section 2 have been implemented in our RSVP engine [3]. In this section, we present and discuss the consequences of the proposed RSVP extensions. This investigation is focused on performance-related aspects.

3.1 Remote Clients

In order to evaluate the remote client extension, there is not much virtue in running large scale performance experiments, because in reality, a first-hop RSVP node is less likely to be challenged by requests from a lot of clients. In general, the number of sessions that

can be handled with this implementation can be estimated to be in the same order of magnitude than what can be sustained at a regular router. It is more interesting to study the effects of the remote client extensions on actual client applications. We look at two interesting numbers, which give an indication that the usage of the remote client API probably does not constitute a severe difficulty, even on small embedded systems. We have taken a very simple rate-based UDP sender and compiled it with and without using the remote RSVP API. The library has been statically linked and we report the size of executables as well as the size of memory allocation for various platforms.

Table 1. Size of Client's Executables and Memory Allocation in Bytes

Platform	Linux 2.2 (Intel)		FreeBSD 3.4 (Intel)		Solaris 2.6 (Sparc)	
Memory Type	Footprint	Data	Footprint	Data	Footprint	Data
with RSVP	96532	980K	171912	1180K	268736	1832K
without RSVP	12488	904K	83368	1016K	141736	1648K
Delta (RSVP)	84044	76K	88544	164K	127000	184K

These results listed in Table 1 remain to be interpreted in the context of real embedded systems, but bearing in mind that the example client is a very simple program consisting of less than 300 lines of code, it can be concluded from these numbers that the increase in executable size and memory allocation due to enabling RSVP capability does not seem prohibitively expensive.

3.2 One-Pass RSVP Signalling

In this section, we report a series of experiments comparing the performance of traditional RSVP signalling with one-pass signalling. Because our RSVP implementation is continually worked on and improved, we report new numbers for traditional signalling, instead of taking them from [4]. All experiments are carried out in the same environment as reported in [4], namely a topology of 450MHz standard Pentium III based PCs running FreeBSD 3.4. For all experiments, we generate a number of sessions and then periodically create and delete sessions in order to simulate an average lifetime of 4 minutes. In all experiments, we report the worst-case CPU processing load and memory allocation at intermediate nodes. Each experiment has run for several minutes and the CPU load number has always stabilized around a value smaller than the peak load. There are no memory leaks in our software, such that the memory allocation remains stable for a given number of flows, as well.

The performance figures for traditional RSVP signalling can be found in Table 2. Although there are slight differences to the earlier numbers reported in [4], it can be concluded that the results are quite similar in their essence. The main difference is given by a decreased variable memory allocation per flow of approximately 1450 bytes, compared to approximately 1850 bytes reported in [4]. In order to evaluate the one-pass reservation mechanism, the same experiment has been run, but employing the one-pass reservation scheme. The results are given in Table 2, as well.

Table 2. Performance of Traditional and One-Pass Signalling

Experiment Settings		Traditional Signalling		One-Pass Signalling	
Number of Flows	Average Lifetime	Load (% CPU)	Memory (in KB)	Load (% CPU)	Memory (in KB)
0	--	0.00	2932	0.00	3004
20000	240 sec	24.56	31628	16.70	27288
40000	240 sec	49.56	60340	34.18	51588
60000	240 sec	74.56	89060	52.25	75888
80000	240 sec	--	--	70.17	100188

Although the implementation has not been optimized for one-pass reservations, at all, a significant improvement of the overall performance is visible. This can be explained mainly by the lower amount of messages that are transmitted. The performance of one-pass signalling is linear to the number flows, as expected, and the memory usage is decreased by more than 200 bytes per flow, compared to traditional signalling. This result is definitely promising with respect to further consideration and potential optimization of this mechanism.

4 Conclusions and Future Work

In this paper, we have evaluated two experimental extensions to RSVP. These extensions are targeted at different scenarios, in which the current specification of RSVP does not provide an adequate set of functionality. The extensions have been implemented and tested to investigate their effect on RSVP's implementation and processing effort. It turns out that the extensions can be realized and used with acceptable effort.

Since the eventual goal of this work is to investigate and design a flexible QoS signalling suite, much additional work remains to be carried out. There are plenty of other potential protocol mechanisms, for example in the field of reservation aggregation. By experimental combination of such mechanisms in a common framework set by our initial RSVP implementation, we hope to gain further insight towards the goal of designing a flexible and modular signalling protocol suite.

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. KOM RSVP Engine, 2001. http://www.kom.e-technik.tu-darmstadt.de/rsvp/.

[4] M. Karsten, J. Schmitt, and R. Steinmetz. Implementation and Evaluation of the KOM RSVP Engine. In *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'2001)*. IEEE, April 2001.