RSVP as Firewall Signalling Protocol

Utz Roedig, Manuel Görtz, Martin Karsten, Ralf Steinmetz
{utz.roedig|manuel.goertz|martin.karsten|ralf.steinmetz}@KOM.tu-darmstadt.de

***Abstract* -- Within a global networked environment, security aspects have become more and more important and access control at network borders is considered essential. For this purpose firewall systems are used which provide a well-established security mechanism to restrict the exchanged traffic to a certain subset of users and applications. In order to cope with the increasing demand for new applications, a firewall must be flexible and extensible to support such new applications and their protocols. RSVP is a dynamic signalling protocol, which has been invented to negotiate resource requirements between end systems and a packet-based communication network. In this paper, we investigate the interoperation of RSVP with a firewall system in order to support new applications in a generic way. We show how the resulting system flexibility allows for a variety of employment scenarios and incremental deployment of such a technology. We back up our claims by describing a prototype that we have implemented.**

I. Introduction

Firewalls are a well established security mechanism for providing access control and auditing at the border between different administrative network domains. To provide its features, the firewall must be compatible with different applications that exchange data between these domains. If a new application is introduced, the firewall must be adapted (by software and/or hardware modifications) to its behavior and its protocols. Therefore a standardized method to add support for new protocols within firewalls in a generic way is absolutely necessary.

The Resource ReSerVation Protocol (RSVP) is designed and used to carry reservation requests for packet-based network protocols. These reservation requests are performed to request a certain Quality of Service (QoS) within the network for an application. The QoS parameters that are normally implemented through RSVP are bandwidth requirements or maximum transmission delays, but it can also be used to signal any other needs that an application has to the network. The appropriate handling of an application within a firewall is also a request that can be signalled by RSVP.

In this paper, we show that RSVP can be used as a firewall signalling protocol. By employing RSVP, an already existing infrastructure can be used to provide a generic framework for interoperation between applications and firewalls.

A. Firewalls

A firewall examines all network traffic between the connected networks. Only data that is explicitly allowed to, as specified by a security policy [1], is able to pass through [2],[3]. In addition to the inspection of data flows, some firewalls also hide the internal network structure of an organization. From the Internet, the only visible and therefore attackable network system is the firewall. This is achieved by the use of Network Address Translation (NAT) [4] mechanisms.

The appropriate treatment of the data that is demanded by the security policy, depend on the application and/or the protocols that are used. Therefore, the firewall has to be instructed on how the data has to be treated for each application. Two general problem domains have to be solved for each application, according to the basic functionality of a firewall:

- Opening and closing paths through the firewall based on security checks.
- Appropriate implementation of the NAT functionality.

The tasks of a firewall are well defined, but there are many possible firewall architectures to fulfil them. Firewalls may consist of packet filters, “stateful filters”, proxies or a combination of all these components. In addition, the applications itself may interact explicitly with a firewall to support the firewall to fulfil their tasks.

B. RSVP

RSVP, initially designed and described in [5], has been specified by the IETF [6] to carry reservation requests for packet-based, stateless network protocols such as IP (Internet Protocol).

In the RSVP model, senders inform RSVP-capable routers and receivers about the possibility for reservation-based communication by advertising their services via PATH messages (Figure 1).

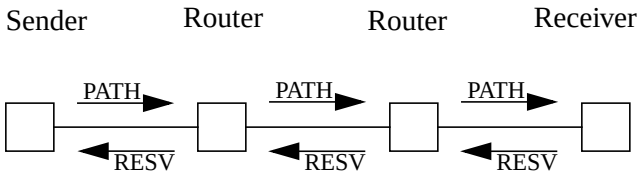

Figure 1 - RSVP session establishment

PATH messages carry the sender's traffic specification (TSpec) and follow exactly the same path towards receivers as data packets. Receivers initiate reservations by replying with RESV messages. They contain a TSpec and a reservation specification (RSpec) and establish the reservation.

C. Motivation and Outline

An explicit interaction between applications and firewalls is necessary, when a general method to add support for new applications within a firewall needs to be available. Especially for the integration of complex and dynamic applications, like multimedia applications, such a general method is desired. To build the interaction between the application and the firewall normally a "firewall framework" is used. In this paper, we show that the existing RSVP architecture can be used to build such a firewall framework. The resulting firewall framework has not to be defined and implemented from scratch, but instead a lot of already existing work can be reused. To explain this fact in detail, we give a general model of a firewall framework. This allows us to show which parts of a firewall framework are already covered by RSVP and which parts have to be added or modified. As we show, modifications of RSVP are not necessary for basic scenarios. RSVP can be "used" as firewall framework.

The next section gives a definition of a firewall framework. In Section III, we show how RSVP is used to build such a firewall framework and we describe its implementation in detail. Afterwards, we discuss the necessary boundary condition, the security of RSVP itself. In Section V, we clarify the functionality of our approach by describing the overall signalling process on the basis of an example application. In Section VI, we present our prototype software implementation. Section VII gives an overview of related work and links our new approach to previous work. We close the paper with a summary of the work and a description of future work that is necessary.

II. Firewall Architectures

Before we define how RSVP can be used as generic firewall framework, we clarify the difference between a firewall using a framework and a firewall that does not. Then we identify which parts are necessary within such a firewall framework in general.

A. Definition of a Basic Firewall

A firewall has to support a set of basic tasks, to properly handle application communication. These basic tasks can be classified in the following manner:

- Task 1. Opening and closing paths through the firewall based on (security) checks.
  - Task 1.1 The firewall checks which hosts are involved in the communication (e.g. by determining source and destination IP-addresses).
  - Task 1.2 The firewall has to determine the service used (e.g. by analyzing the UDP or TCP ports).
  - Task 1.3 The firewall has to know about the communication dependencies and has to adjust its configuration dynamically (e.g. relationship of several flows that form one session).
  - Task 1.4 The firewall determines the users that are involved in the communication (e.g. by authentification of a user within a proxy).
- Task 2. Implementation of the NAT functionality
  - Task 2.1 NAT is implemented by modifying the IP-Addresses in layer 3 and for TCP and UDP, the ports in layer 4.
  - Task 2.2 For some applications, modifications of the higher layers are necessary. (e.g if IP-addresses are negotiated and submitted in high layers during communication).

In the next sections, we use the term *basic firewall* to refer to a firewall or firewall system which is capable to execute the tasks listed above. Not all of these tasks are performed by all types of firewall components (filters, stateful filters, proxies). For example a packet filter does not support the Task 1.4 and NAT. Standard firewalls, e.g. *Firewall-1* [7] or *Gauntlet* [8], however are able to execute all of the mentioned tasks. Some of these firewalls might provide additional tasks (e.g. virus checking) but these are not included in our definition of a *basic firewall* (see Section A). Every firewall, using a framework or not, should support the tasks of a *basic firewall*.

B. Grouping of Logical Parts

For some of the described tasks of a *basic firewall* a firewall needs some information, that is normally not available to network nodes. Therefore, the firewall has to interact with the application or its protocols. Figure 2 shows the *logical parts* that can be grouped to build different levels of interaction between application and firewall.

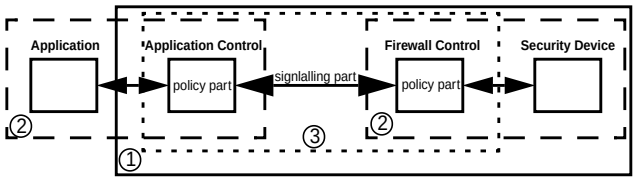

Figure 2 - logical parts of a firewall

The part *Application Control* is responsible to extract the information (e.g. owner of the data) from the applications communication that is necessary for the firewall. This information are then passed to the *Firewall Control*. The *Firewall Control* is responsible to handover this information to the security device (e.g. a packet filter device). The appearance of this information depend on the used security device type (e.g. commands to configure a specific packet filter device).

Combination ① in Figure 2 shows a "state of the art" firewall, where the application does not interact directly with the firewall to help the firewall performing its tasks. In this case, all logical parts are located on the firewall itself. The *Application Control* is designed as protocol parser within the firewall. The information extracted by the *Application Control* is then passed via function call to the *Firewall Control* which is also located on the firewall. The *Firewall Control* then executes the appropriate firewall configuration commands.

When a firewall framework ② is used, the *Application Control* is part of the application. The necessary information is passed by the application to the application control, so that a parser is not necessary. Then the information is passed to the *Firewall Control* which is located on the firewall. Since the application is running on a different machine than the firewall, a mechanism to transport the information over the network is needed. The *Firewall Control* then executes the appropriate firewall configuration commands.

It is also possible to group the logical parts in other ways. Combination ③ in Figure 2 shows an intermediate proxy between the application and the firewall.

The main difference between combination ① and ② is given by the generality of the approaches. In the first approach, the firewall needs to interoperate with a parser. This parser must have knowledge of the internal structure of the application protocols, and therefore a parser for each application is necessary.

The first approach could be considered to be more secure, due to the granularity of the possible checks in the specialized parser and the missing interaction with other network components (see Section C). In the second approach, the firewall support has to be integrated in the application, but the firewall has not to be modified for every new application. There is a trade off between security and generality, which has to be considered.

A firewall framework consists of the following two main parts:

- The transportation of the necessary information between applications and the firewall. We refer to this part of a framework as the *signalling-part.*
- The definition, how these informations are created in the applications (*Application Control*) and how these informations are used in the firewall (*Firewall Control*). We refer to this part of the framework as *policy-part.*

Both parts have to interact in a way that the resulting framework can handle the tasks of a *basic firewall*. The design of each part has an impact on the design of the other part.

C. Security Concerns

If a firewall framework is used to replace specialized firewall components, like an application level gateway, there is some functionality that could not be covered. An application level gateway or proxy compromises specialized parsers for the supported applications. These parsers are capable to perform security checks on the application layer. For example an FTP application level gateway can filter certain ftp commands like GET or PUT, an HTTP application level gateway can remove ActiveX or java applications from the transferred documents. There is a trade-off between "extended security features" and "general application support".

III. An RSVP Based Firewall Framework

Firewalls can use RSVP in two different ways. First, a firewall is also a router and therefore could use RSVP in the same way as normal QoS aware routers. In case that the flow's packets are not simply forwarded but checked by some application code on the firewall, the resources to be reserved are not only network resources but local components as well. Hence, mechanisms such as CPU scheduling and memory management must be considered [9]. This usage of RSVP is related to classical QoS which also considers the specific behavior of a security component. The details of network and local resource management are out of the scope of this paper.

The second method, how a firewall can use RSVP, is to use it as signalling protocol for security QoS requests. A desired security level can be part of a QoS specification [9] as well. In the literature such a security QoS is sometimes introduced as Quality of Protection QoP [10]. This method of using RSVP is discussed within this paper.

A. Definition

To avoid the need of implementing and installing a new parser everytime a new protocol has to cooperate with the firewall, we propose the "Firewall RSVP Framework". We suggest to use RSVP to transport the necessary information between applications and the firewall. RSVP is used in this framework to implement the *signalling part* and the *policy part*.

The application has to announce every upcoming stream by sending a standard RSVP PATH message which represents the *application control* in the *policy-part*. The message contains a description of the upcoming flow, e.g. protocol, source, destination. These information are used by the firewall to implement the appropriate mechanisms, representing the *firewall control* within the *policy-part*. Because these steps are performed for each flow, the firewall does not have to analyze flow dependencies or flow characteristics of an application session to support the tasks 1.1 to 1.3. Thereby, the *signalling-part* uses the standard RSVP signalling mechanisms, as they are also used for QoS purpose. We define this operation mode as the *passive mode* where existing information in the RSVP messages is interpreted for security purposes within the firewall. To be able to fulfil also the necessary Task 1.4, the signalling part has to use an *active mode*. Within the RSVP PATH message an additional object has to be inserted by the application, which contains information about the flow's owner. This can be achieved by using the already defined POLICY_DATA-object.

To match the NAT requirements specified in task 2. (Section II.A), the *active mode* might be used as well, but this is not discussed within this paper.

As we have shown, two "operation modes" have to be distinguished. Both modes can be used within a RSVP firewall framework. When the *passive mode* is used, the standard RSVP signalling can be employed to control a firewall supporting Tasks 1.1 to 1.3. To control a firewall which supports all tasks of a basic firewall, the *active mode* has to be used. To use the active mode, the RSVP signalling has to be enhanced which is well feasible due to the modular design of RSVP.

This paper investigates both operation modes. We show how these two operation modes can be used to build a generic framework for the integration of new applications within firewalls. Thereby, we primarily focus on the firewall problem domain *"Opening and closing paths"*. The problem domain *"NAT functionality"* is considered but not be discussed in detail in this paper.

B. Preconditions

If RSVP is used as a firewall framework, some preconditions have to be considered.

At least two of the involved nodes (one communication endpoint and the firewall) have to support the RSVP protocol. Compared to other frameworks, as described above, this results in the same implementation effort, but also additional features are available.

If external entities (applications, devices,...) can influence the configuration of the firewall, a *policy overlay* is necessary. This means that an administrator of the firewall should be able to narrow the possible configuration changes that are signalled. For example, an administrator wants to be sure that negotiated flows are only enabled when they use not well-known ports. This ensures that even in a worst case scenario a basic firewall remains active.

If RSVP is transporting security related information, which is true when the firewall configuration is based on the RSVP message content, it has to ensure that the protocol itself is secure. Because this is the most important precondition, we discuss this issue in detail in section IV.

When these preconditions are fulfilled, the following mechanisms have to be implemented to enable RSVP for the targeted purpose.

C. Details

Figure 3 describes a simple scenario where an intranet is protected by a firewall which uses the RSVP framework. We refer to this scenario to describe the details of the framework.

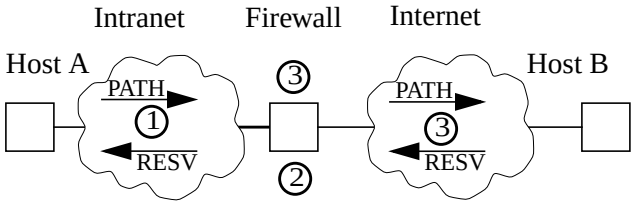

Figure 3 - RSVP session establishment

If Host A intends to initiate a flow to Host B it announces this upcoming flow by sending a PATH message to HOST B (①). The firewall, which is capable to take part in the RSVP signalling, processes and forwards the RSVP information to host B. Host B replies with a

RESV message (③). This message is also processed by the firewall and forwarded to host A.

1) Policy Part - Application Control

The applications have to announce all flows that are used by RSVP. Therefore each application has to use the standard RSVP mechanisms. If the firewall also requires user authentification for the flows, the RSVP implementation also has to support the POLICY_DATA-object as described in [11].

Applications have to implement a standard RSVP signalling interface to provide the *Application Control*.

2) Policy Part - Firewall Control

To implement this part of the framework, the firewall has to take part in RSVP signalling. This could be achieved by adding a standard RSVP daemon like [12] or [13] to the firewall. The used RSVP daemon has to be slightly modified to be able to support the following capabilities:

If host A initiates the communication (as described above), the firewall uses the information in the PATH message to implement a filter rule for the upcoming flow. If host B initiates the communication, host B sends the PATH message and host A responds with the RESV message. In this case the information within the RESV message is used by the firewall to implement the appropriate filter rule. As described in section IV, security mechanisms can be used to authenticate the generator of RSVP messages. Because of the hop by hop characteristic of RSVP, also the security mechanisms are hop based. Therefore the firewall can only trust RSVP messages that

- are passed by the host directly without passing an RSVP hop and
- are passed from a hop that is trusted by the firewall.

These conditions can only by satisfied by signed messages, containing firewall signalling information, from internal and therefore trusted hosts.

Consequently either the PATH or the RESV message are used to update the firewall configuration, depending on who is the internal sender. A *Connection-Cache* is necessary in the firewall, to control if the internal host has confirmed the communication request. The *Connection-Cache* is a table which holds entries for each connection. It stores source and destination addresses and ports as well as the used protocol, the direction (internal or external) and the status of the connection. The RSVP implementation within the firewall has to distinguish internal and external generated messages

An RSVP message (PATH and RESV), generated by the application to announce an upcoming flow, usually has the appearance shown in Figure 4.

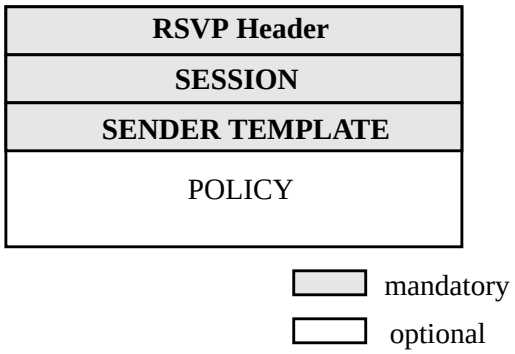

Figure 4 - RSVP message

The protocol number, the destination port and destination address are included in the SESSION object. The source port and source address are included in the SENDER_TEMPLATE object. Information about the user that is responsible for the flow is included in the POLICY_DATA object. The usage of the POLICY_-DATA object is currently not available in most RSVP implementations, but it is defined within the RSVP standards. The necessary information is extracted from the RSVP message and processed by the firewall in the following manner:

- The information about the upcoming flow are extracted from the RSVP message.
- The informations are compared with the firewall security policy. The firewall checks, if the desired communication endpoints are valid and the user is authorized for communication. (Task 1.1,1.2,1.4 of the *basic firewall*. Task 1.3 is not performed by the firewall, this task is now performed by the application).
- The firewall checks whether the negotiated communication parameters match with the *policy overlay.*
- The firewall creates the filter rules using the information from the RSVP packets. Depending on the used firewall type, several rules might be necessary to enable one flow.

Rules that are not needed anymore have to be removed properly. RSVP provides a soft state characteristic. The applications have to take care of the implemented QoS reservations. They have to refresh their reservations periodically by sending appropriate messages. If the refresh messages are not sent, the implemented reservations in the network nodes are removed. This feature is also used within the firewall to remove rules that are not needed anymore. Alternatively applications might send a TEAR message to remove the implemented QoS parameters

explicitly. In this case, the TEAR message is used by the firewall to remove the rules.

3) Signalling Part

The *signalling part* is implemented by using an RSVP implementation (in the application and on the firewall) that supports basic security as defined in Section IV. The *signalling part* uses the standard RSVP signalling mechanisms.

D. Limitations and open Questions

The additional signalling between applications and the firewall brings up performance questions. It is shown in [14] that the RSVP signalling is performant enough to be used even for large number of concurrent sessions.

This paper does not cover the problem domain "NAT functionality", which has to be considered. For NAT functions, additional RSVP objects have to be designed to signal appropriate information between application and firewall.

There are also other fields which have to be investigated, e.g. multicast, real-time operation, extensibility.

IV. Security Mechanisms in RSVP

Security can be divided into three major issues that can be distinguished: *Authentication, Integrity,* and *Confidentiality.* For all three goals numerous approaches exist on the market today and there are also solutions for the interworking with RSVP.

A. Authentication

The identification of a communication partner is independent from the actual message and is based upon cryptographic methods. In [15] an identity representation for RSVP is introduced. The required information to support policy based admission control is placed in POLICY_DATA objects, which itself consists of other objects which depend on the used identification mechanism, e.g. Kerberos [16] or User Distinguished Names. A user process inserts its created authentication data into the RSVP message and sends it to the next hop, where the message is authenticated and a new authentication is inserted. The authentication scheme in RSVP is based on hop-by-hop, like other methods too.

B. Integrity

Beside knowing that the sender of the message or the last hop respectively can be trusted, it is important that the content of the message is not corrupted. To ensure the integrity of the message a cryptographic hashfunction in conjunction with a message digest (HMAC-M5) is used. In [17] the optional INTEGRITY object is defined, which consists of three main fields

- Key Identifier – a unique number for a given sender.
- Sequence Number – a one-time-use 64-bit monotonically increasing number to thwart replay attacks. This sequence number is determined by a challenge response method within RSVP.
- Keyed Message Digest – the calculated authentication information for this message.

The INTEGRITY object must be the first object after the RSVP-header. The next capable hop selects the key based on the Key Identifier and the senders IP address and verifies that the integrity of the message is not violated. If the verification fails, the message is dropped otherwise a new message digest will be created from the current system. The end-to-end authentication of the message between sender and receiver can be provided only by a chain of trusted hops.

C. Confidentiality

To retain confidentiality between the communication parties it is necessary to encrypt the body of the message, for instance using IPsec [18] [19]. IPsec can be used in conjunction with RSVP to provide confidentiality. In addition this approach is an alternative for authentication and integrity.

D. Summary

All three aspects of security can be covered by methods for RSVP. The basic idea behind most methods is the transformation from end-to-end concepts to a hop-by-hop basis, which fits better into the RSVP environment.

Authentication and Integrity must be provided to be able to perform the *basic firewall* Task 1.4 (Section II.A) to ensure that the signalling messages are not modified during the transport. Confidentiality is not necessary, because an intruder can either look at the RSVP message content or the subsequent communication. Communication relations can be determined by an intruder anyway.

Therefore, IPsec is not necessary, because the security requirements can be met by using two additional objects POLICY_DATA and INTEGRITY.

If these objects are used, RSVP can be considered secure and capable to carry information to configure a firewall.

V. Communication Examples

To make the previous described concept more vivid, a real-world examples is explained in this section. In the chosen example, *RealAudio* reflects a currently often used application. The RealPlayer from RealNetworks is presently a widespread application for streamed audio and video over the Internet. This application has a dynamic behavior in negotiating the data connections and is complex to handle for the firewall.

The signalling between a client, the firewall and a server for a RealAudio communication is shown in Figure 5 and described below. The communication starts with an RTSP [20] connection from the client to the well-known port (554) on the server. This connection is used to negotiate parameters, e.g. ports, for the upcoming media data transmission using RTP [21] over UDP. To provide synchronization information to the client a full-duplex UDP channel is opened for RTCP. The choice of the port number for the RTP channel is only limited by the rule that it must be an even number above 1023 and for RTCP connection the port must be the next consecutive one. For an administrator of a firewall system it is a priori not known what port for the media data delivery will be chosen. Hence, a dynamic ruleset has to be created to allow RTP streams negotiated by RTSP through the firewall. Applying filter rules for the current session on-the-fly is the only solution, because opening all ports greater 1023 is no alternative.

With our suggestion to use RSVP for signalling, the following situation arise:

- The receiver (here: RealPlayer-client) establishes a RTSP connection via TCP to the server through the firewall, which is not a problem because the connection is to a well-known port and predefined rules can be used.
- After negotiating the parameters for the media streams, the sender (here: RealAudio-Server) initiates the data transmission and the control flow (RTCP). For this purpose the sender sends to PATH messages, one for each flow, to the receiver.
- The informations in the PATH messages are inserted in the *ConnectionCache*, because the sender is an external sender and therefor untrusted.
- The receiver replies with two RESV messages.
- The firewall receives the RSVP message from the RSVP engine and extracts sender and receiver port and address and the underlying protocol from the objects. If the POLICY_DATA object is present, the user can be identified as well.
- The integrity of the message is checked as well as the authentication. If the verification succeeds, a new rule is created and the parameters for the connection are updated in the *ConnectionCache*.
- The arriving UDP stream matches now the criteria in the rulesets from the firewall. The same is true for the RTCP stream.
- After closing the RealAudio session, the specific firewall rules time out or are removed by respective TEAR messages.

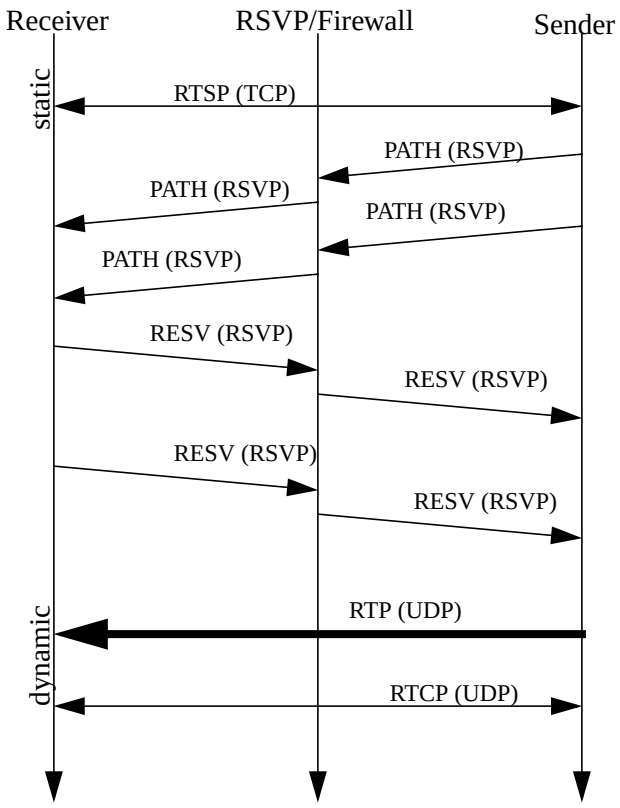

Figure 5 - RealAudio signalling

The use of RSVP allows to get all necessary information for the creation, installation and deletion of temporary firewall rules.

VI. Implementation

To evaluate the concept of using RSVP messages to signal information to a firewall, a prototype implementation has been developed.

The implementation only handles a few functions, that can fulfill the basic tasks for the "*Opening and Closing path*" problem domain like *creating* and *deleting* rules for the packet filter.

The model of a RSVP engine is taken from [6] and shown in Figure 6. For each outgoing interface, the *Packet Scheduler* is responsible for achieving the promised QoS. All message types influencing the QoS like PATH and RESV are handled by this module, as well as the TEAR messages.

Due to the fact, that these messages carry all the important signalling informations for the firewall, the *Packet Scheduler* is expanded with new functions to ensure the communication with the firewall.

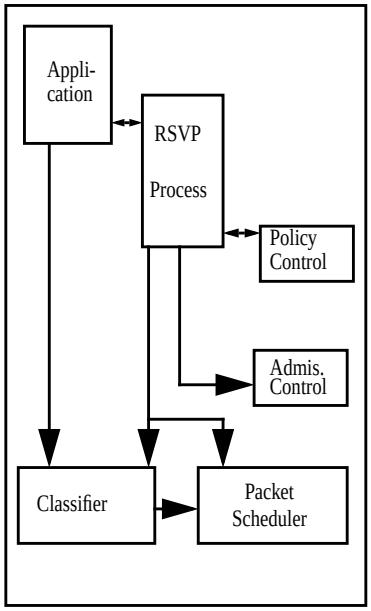

Figure 6 - Modules and flows inside a RSVP host

Below the part of the RSVP engine, which is important for communication with the firewall is described in detail. The described classes refer to the RSVP implementation described in [13].

A. Class hierarchy

The class hierarchy of *Scheduler* and its derived classes is shown in Figure 8. The class *Scheduler* acts as a base class for different flavors of scheduling packages and provides a common interface to them. It provides the methods shown in Figure 7, needed to perform the requests transported by RSVP.

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

Figure 7 - Main methods in Scheduler

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 which are not shown here.

The class *Scheduler* which represents the module *Packet Scheduler* in Figure 6 is used as an entry point for the communication between the RSVP engine and the firewall. From this class a new class named *SchedulerFW* is derived and the necessary functions for the interoperation with the firewall are replaced by customized ones.

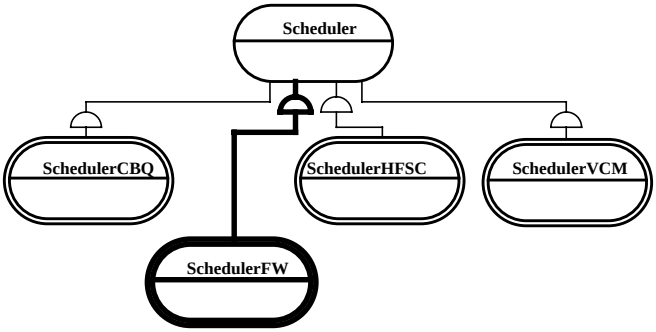

Figure 8 - Class hierarchy in the KOM-RSVP-Engine

B. SchedulerFW

In the class *SchedulerFW* the methods for adding, deleting and modifying filters are overloaded to accomplish the functions needed to operate a firewall. The method receives as parameters primarily the *Sender*- and *Session*-objects.

These contain, as shown in this paper, all necessary information to describe a connection. The sender and receiver address and port as well as the protocol id can be extracted from these objects and used for building new temporary filter rules in the firewall. These new filter rules are created inside *SchedulerFW* with respect to the filter structure of the particular packet-filter [22].

The removement of the rules is pushed by expiring of the softstate in RSVP itself or by explicit TEAR messages. After processing these messages, another system-call removes the rule from the firewall. Both, the firewall and the RSVP engine, are currently located on one machine.

VII. Related Work

There are some frameworks available, that can be used to handle applications within a firewall in a generic way. New frameworks are currently investigated and proposed in the standardization organizations (IETF and ITU). Well known methods can be used to solve the described problems for standard applications (e.g. like Telnet or Http). New and currently discussed methods are necessary, because the requirements of new applications (e.g. for IP-Telephony or Video streaming) can not be fulfilled by the existing frameworks [23]. This is caused by the special characteristics of these applications [24]. Most of

the proposed frameworks try to cover both firewall problem domains (*"Opening and closing paths"* and *"NAT functionality")*, which makes sense because these problem domains mostly show up together and are related to the same physical device.

One firewall framework is the SOCKS approach, specified by the Authenticated Firewall Traversal Working Group [25]. Applications have to be recompiled with an appropriate SOCKS library, then the usage of a firewall is possible in a standardized way. Thereby both firewall problem domains are addressed. TCP and UDP based communication is supported by the socks framework, but UDP is not handled in an efficient way [23]. TCP connections are used to inform the firewall about the UDP streams, therefore one TCP control connection for every UDP stream is used. With such an approach, new application types like multimedia applications could neither be supported in a efficient way nor for a large number of connections.

A new framework is proposed by the IETF, which defines a signalling protocol. The applications inform the firewall about their needs through a Firewall Control Protocol (FCP) [26]. This draft covers both firewall problem domains. The application type which initially is targeted by this work, is IP-Telephony based on SIP. Therefore, this framework is optimized for multimedia applications, and their specific needs are considered. Up to now, the draft only defines the general requirements for the FCP. Neither the exact design of the FCP is given, nor the transport protocol that should carry the FCP messages.

A second IETF draft [27] addresses only the IP-Telephony H.323 firewall problem. NAT or protocols other than H.323 are not considered. In this draft, a H.323 Firewall Control Protocol (HFCP) is proposed. Technically it is the same approach as the previous one, described in more detail but only for H.323 applications in non NAT environments.

The ITU is proposing a VoIP Firewall Control Protocol [28]. This protocol targets all firewall problem domains and the protocol can also be used for all kind of applications. The protocol itself is, as the name of the protocol indicates, optimized for IP-Telephony. This protocol is specified in detail and first implementations exist. As a carrier for the FCP a RPC like mechanism is proposed.

Our approach is also based on a signalling between the communication endpoints (applications) and the firewall. The differences between our approach and the described existing solutions are the following. An existing protocol is used, namely RSVP. In the *passive mode*, information that is already available is used. No additional effort at the client side is necessary to solve the firewall problem domain *"Opening and closing paths"*. Our approach is more general, because it is not application specific and can deal with "modern" applications. Our approach is more integrating, because two related problems can be solved (QoS and QoP) using one framework.

VIII. Conclusion

We have shown that RSVP can be used as generic firewall framework. Compared to other existing frameworks this approach is based on already established and used methods. By combining both approaches, RSVP signalling and explicit firewall control, both benefit. Firewalls will benefit if they are able to provide a generic interface, that is well known and is already accepted by application developers. RSVP benefits by providing an additional feature and becoming thereby a more integrating signalling approach.

We believe that in the near future explicit service negotiation will be used on the network and that RSVP will be used for this purpose. Security can be seen as a QoS parameter (QoP) and it is sound to use the same protocol to signal all QoS parameters.

We have shown that RSVP already provides a lot of mechanisms that are also necessary to build a firewall framework. Few extensions to RSVP are necessary to also cover the task "firewall control".

References

[1] D. Curry: UNIX System Security, Addison-Wesley, 1995

[2] D. Chapman. Building Internet Firewalls,O'Reilly, Cambridge, 1995

[3] W. Cheswick and S. Bellovin. Firewalls and Internet Security, Addison Wesley, 1994

[4] K. Evang and P. Franci. RFC 1631: The IP Network Address Translator (NAT) Status: INFORMATIONAL, 1994

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

[6] 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.

[7] Checkpoint Software Technologies Ltd. Firewall-1. http://www.checkpoint.com

[8] Network Associates Inc. Gauntlet Firewall. http://www.pgp.com/products/gauntlet/firewall

[9] R. Steinmetz and L. Wolf. Quality of Service: Where are We?. In Proceedings of the IFIP Fifth International Workshop on Quality of Service (IWQOS '97), New York, NY, USA, May 1997.

[10] Object Management Group: Quality of Service OMG Green Paper, June 1997.

[11] S. Herzog. RFC 2750: RSVP Extensions for Policy Control. Standards Track RFC, January 2000

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

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

[14] 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. To appear.

[15] S. Yadav, R. Yavatkar, R. Pabbati, P. Ford, T. Moore and S. Herzoh. RFC 2752: Identity Representation for RSVP. Standards Track RFC, January 2000

[16] S.P. Miller, B.C. Neuman, J.I. Schiller and J.H. Saltzer. Section E.2.1.: Kerberos Authentication and Authorization System. MIT Project Athena, Dec 1987.

[17] F. Baker, B. Lindell and M. Talwar. RFC 2747: RSVP Cryptographic Authentication. Standards Track RFC, January 2000

[18] R. Atkinson. RFC 1826: IP Authentication Header. Standards Track RFC, August 1995

[19] R. Atkinson. RFC 1826: IP Encapsulating Security Payload (ESP). Standards Track, August 1995

[20] H. Schulzrinne, A. Rao, R. Lanphier. RFC 2326: Real Time Streaming Protocol (RTSP). Standards Track RFC, April 1998

[21] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson. RFC 1889: RTP: A Transport Protocol for Real-Time Applications. Standards Track RFC, January 1996.

[22] D. Reed. IP-Filter - TCP/IP Packet Filtering Package. http://coombs.anu.edu.au/~avalon/ip-filter.htm, 2000

[23] A.Molitor. Firewall Control for IP Telephony. ARAVOX Technologies. http://www.aravox.com.

[24] U. Roedig, R. Ackermann, C. Rensing, and R. Steinmetz. A Distributed Firewall for Multimedia Applications. In Proceedings of the Workshop "Sicherheit in Mediendaten", September 2000.

[25] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas and L. Jones. RFC 1928: SOCKS Protocol Version 5, Standards Track RFC, March 1996.

[26] J. Kuthan and J. Rosenberg. Firewall Control Protocol Framework and Requirements. draft-kuthan-fcp-01.txt, June 2000

[27] S. Mercer, A. Moilitor, M. Hurry and T. Ngo. Internet Draft draft-rfced-inf-mercer-00.txt, H.323 Firewall Control Interface (HFCI). June 1999.

[28] TIPHON Working Group.VoIP Firewall Control Protocol (FCP), 2000. http://www.etsi.org/tiphon/