An Alternative Approach to RSVP-aware MBone Applications

**Yu-Ben Miao, **Chen-Yu Wang, **Ji-Feng Chiu, *Wen-Shyang Hwang, **Ce-Kuen Shieh
*Department of Electrical Engineering, National Kaohsiung University of Applied Sciences, Taiwan, R.O.C.
**Department of Electrical Engineering, National Cheng Kung University, Taiwan, R.O.C.
ybmiau@ybmiau.ee.ncku.edu.tw, aries@kungsrv.ee.ncku.edu.tw, gary@hpds.ee.ncku.edu.tw,
wshwang@mail.ee.kuas.edu.tw, shieh@eembox.ee.ncku.edu.tw

Abstract

QoS guarantee is one of the most critical issues for deploying multicast-style applications. However, the complexity of communication between applications and QoS mechanism decreases the utility of QoS provisioning. In this paper, we will propose an alternative approach to RSVP-aware MBone applications. This approach adopts RLR (RSVP Library Redirection), which can transform legacy Internet applications into RSVP-aware without modifying their source codes. We will extend the ability of RLR to support UDP (User Datagram Protocol), multicast-style MBone applications and use Vic as an example to illustrate how to transform legacy applications into RSVP-aware. We also deploy a user interface for users to adjust their reservation requirement.

I. Introduction

Nowadays, real-time applications such as video conferencing, interactive multimedia games and VOD (Video On Demand), are becoming popular in the Internet. Basically, these applications run on MBone (Multicast Backbone), a network that consists of multicast-capable routers, and are called MBone applications. Most of them are UDP (User Datagram Protocol) -based, multicast-style and QoS-sensitive applications. However, in a best-effort network, there is no guarantee of the network QoS. The packets of UDP-based applications may be dropped or delayed when the network is congested. To solve this problem, QoS-aware network and QoS-aware applications are necessary. Many QoS mechanisms for network such as IntServ (Integrated Service) or IntServ plus DiffServ (Differentiated Service) have been developed extensively in past few years. In order to couple QoS-aware applications with QoS-aware networks, a signaling protocol RSVP (Resource reSerVation Protocol) has been proposed [1][2][7]. However, although the QoS-aware network has been deploying well, the sophisticated implementation of QoS-aware application becomes one of the bottlenecks to utilize QoS-aware network for QoS-sensitive applications.

In [3] we proposed a transparent method named RLR (RSVP Library Redirection) to transform legacy application to be RSVP-aware. RLR achieves the transparent transformation by redirecting procedure calls from the socket library to the RAPI (RSVP Application Programming Interface) library without any modification of source codes. However, the previous study focuses on FTP applications, which are TCP-based, unicast-style. It did not give the solution for UDP-based, multicast-style applications. Furthermore, the parameters that describe the traffic characteristic are preset in the program. These parameters can't be dynamically changed to adapt to variable network situation and application requirements. In addition, it is usually hard for users to specify these parameters precisely. A convenient way for users to choose the quality of service they desired is in demand.

This paper focuses on the RLR approach for UDP-based, multicast-style applications. A typical MBone application *Vic* [6] is chosen to be the target application. *Vic* is a popular shareware on the Internet. It is an UDP-based multicast-style and QoS-sensitive application hence it is suit for verifying the feasibility of RLR. Also a user-friendly interface is developed for user to arbitrarily and dynamically adjusts the reservation requirement.

The rest of this paper organizes as follows: Section 2 introduces the background of RLR, and how multicast applications work. Section 3 gives a description of how to design RSVP-aware multicast applications. Section 4 illustrates the implementation of RSVP-aware MBone applications by using RLR. The experimental results are shown in Section 5. Finally, Section 6 gives a conclusion remark.

II. Background

RLR

RLR is used to transparently transform legacy Internet applications into RSVP-aware by means of library redirection. Fig 1 shows the control flow of RLR.

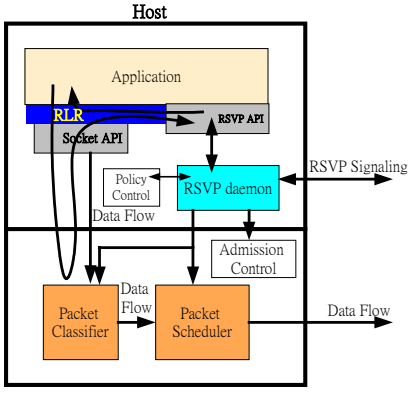


Fig. 1 The control flow of RLR

First, the related invocations of socket routines from applications are intercepted. Second, collecting the parameters included in the intercepted routine calls if necessary. Third, the originally invoked socket routines continue to complete. Fourth, instead of returning directly to applications after completion of the socket routines, the control flow is redirected to call the RAPI to communicate with RSVP daemon. Consequently RSVP daemon will issue associated RSVP signaling messages. Fifth, after finishing the signaling procedures, the control flow is returned to the applications.

MBone applications

MBone has been the testbed for multimedia applications such as audio conferencing tool, video conferencing tool, and many others. Most of these applications work on RTP (Real-time Transport Protocol) [4]. RTP is the protocol that provides end-to-end delivery services for temporally sensitive data. Its data transport is augmented by a control protocol (RTP Control Protocol, RTCP) that allows monitoring of the data delivery by providing feedback on the quality of the data distribution. Usually, both protocols are based on UDP. Although RTP/RTCP supports certain QoS monitoring information, RTP/RTCP itself does not provide any mechanism to ensure timely delivery or provide other quality-of-service guarantees [4]. It relies on the QoS provisioned by the under layer network.

Fig. 2 shows the time line of a typical MBone application that initiates a RTP session within two hosts [5]. An RTP session begins with the contributing source (sender) starting to send a media stream even though there may be no receiver at that time. Some time later, the receiver sends out an IGMP-join packet to join the session and starts to receive the data of this media stream. Both Sender and Receiver periodically send RTCP control messages once they join the RTP session.

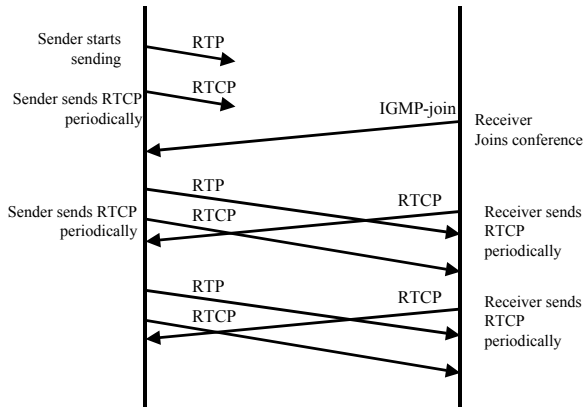


Fig. 2 Time sequence when a session goes

III. Design of RSVP-aware MBone applications

Basically, RSVP-aware applications communicate with RSVP daemon by invoking RAPI function calls to issue related RSVP messages. Sender has to issue PATH message with the parameter that describes the characteristic of traffic it is going to generate. After receiving sender's PATH message, the receiver issues a RESV message with parameter that specifies the resource it requires. However, traditional applications do not deal with the invocation of RAPI functions. Therefore, RLR approach is in charge of invoking proper RAPI function calls for legacy applications to communicate with RSVP daemon. RLR has to identify which traffics need to be protected and make adequate resource reservation.

As mentioned above, there are RTP and RTCP flows in a RTP session that require adequate resource reservation. Fig 3 shows the occasions that a RSVP-aware MBone application sends RSVP messages to reserve resource for RTP flow.

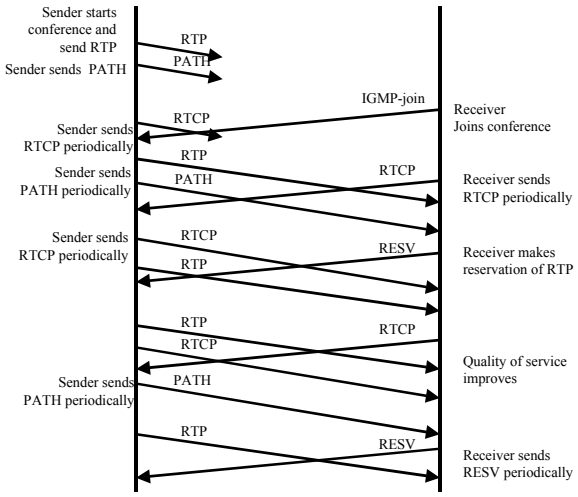


Fig. 3 Make reservation for RTP flow

First, sender begins to send out RSVP PATH messages while it is initiating a RTP session to distribute a media stream. According to the multicast routing tables, the routers of MBone reproduce and forward these PATH messages in the same manner as distributing the RTP packets. Second, receiver issues IGMP-join to participate in this RTP session. Third, routers start to forward the RTP flow and PATH messages to that receiver. Forth, after receiving the PATH message, receiver issues RSVP RESV messages to make reservation for the RTP flow. Fifth, the routers aggregate the RESV messages if necessary and send these messages backward hop by hop to the sender to accomplish the reservation.

The protocol behavior of RTCP is similar to RTP except that both sender and receiver generate RTCP flows. In this case, a bi-directional RSVP reservation is required.

IV. Implementation

RLRs for MBone Applications

To transform *Vic* into RSVP-aware by using RLR approach, it is critical to know their protocol behavior, i.e. the sequence of socket functions invocation. The time of when to invoke RAPI functions must be acknowledged. Also the parameters that required by the RAPI function

call such as IP address and port number need to be collected. The remote IP address and remote port number are available before the application runs since these parameters must be predefined in a multicast scenario. However, most applications bind the local port dynamically. The port number can be retrieved only at runtime, which means some proper socket function calls need to be intercepted to collect the information of local port number. Fig 4 gives the flow of socket function calls when *Vic* runs. There are two pairs of sockets that *Vic* opens. Fig. 4(a) and (c) are the sockets that individually send and receive RTP flow. Fig. 4(b) and (d) are the sockets for RTCP flow. In Fig. 4(a), *Vic* calls *socket()* to open a socket, named S_RTP_Send, for sending RTP payload. Then it calls *connect()* to set remote IP address (a class D group IP address in multicast scenario). V*ic* also sets socket options to set loopback, TTL of RTP packets and send buffer size. Finally V*ic* calls *sendmsg()* to send out RTP packets.

To receive RTP packets, *Vic* open a socket named S_RTP_Receive, see Fig 4(c). It has to call *setsockopt()* to set option SO_REUSEADDR and SO_REUSEPORT. It is common routine for multicast programs to allow multiple instances running simultaneously. Later, it calls *bind()* to bind a local protocol address. To receive packets from the specified group, *Vic* sets option *IP_ADD_MEMBERSHIP* to join the group. After setting receive buffer size, V*ic* calls *recvfrom()* to receive RTP packets.

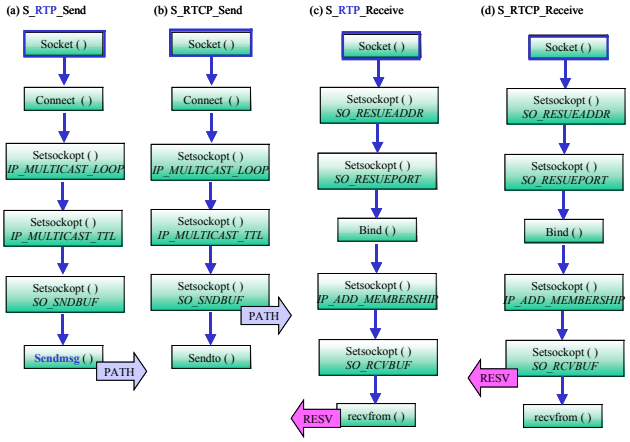


Fig. 4 Flow of socket function calls when *Vic* runs

In a RTP session of *Vic*, the host that plays the contributing source role has to issue PATH message to setup the RSVP path state for the RTP flow. Two steps are needed to do that. First, the *connect()* or *setsockopt()* need to be intercepted to retrieve the group IP address and port number. Second, RLR issues RSVP PATH message while *sendmsg()* is invoked. On the other hand, the host that plays the recipient role has to issue RESV message to make reservation for the RTP flow. In this case, *bind()* or *setsockopt()* are the functions to be intercepted to retrieve the group IP and port. RSVP RESV message is sent at the time when *recvfrom()* is invoked. The similar routines are applied to make reservation for RTCP flow, except that PATH and RESV messages are sent while *setsockopt()* is invoked.

RSVP parameters setting agent

To support convenience for user to specify the reservation specification, a GUI (Graphic User Interface) is introduced in Fig 5. When the legacy application starts with RLR, RLR will launch RSVP parameters setting agent. There is a list maintained by the agent. The agent starts with checking the user ID and consulting the list to get the maximum of token rate, bucket size, and peak rate that the user can specify. After that an interactive window is popped up. User can adjust the reservation parameters by scrolling the track bars. As the "Submit RSVP" button is clicked, the agent sends these parameters through IPC (Inter Process Communication) and signals RLR to retrieve them out. Consequently, RLR invokes related RAPI to issue this new reservation requirement. The results of RAPI invocation will be translate and passed to the agent through IPC. The status bar below the window displays the results to indicate user whether the reservation succeeds or not. When application is running, the user can arbitrarily change the reservation requirements to get a desired quality of service.

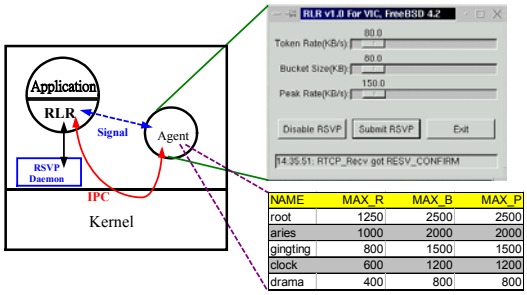


Fig. 5 Design of RSVP Parameters setting Agent

V. Experiments and analysis

To verify the feasibility of RLR approach for multicast-style, QoS-sensitive applications, we set up two experiments. The functionality of RSVP parameters setting agent is also examined. Fig 6 gives the experiments topology. There are seven PCs named CANCER, LEO, ARIES, TAURUS, VIRGO, GEMINI and SCORPIO. Their network interfaces are 10 Mbps Ethernet links and FreeBSD is installed in these PCs.

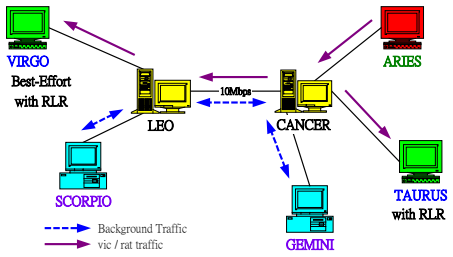


Fig. 6 Experiment topology

CANCER and LEO are used as PC-based routers to connect 2 subnets. ARIES is the contributing source that

distributes media stream in the conference, while TAURUS and VIRGO are conference receivers. Background noise traffic is generated from GEMINI to SCORPIO to simulate varied condition of the network load. Fig 7 shows the link-sharing structure for these experiments. Best-effort service is allocated with 60% bandwidth and RSVP service is assigned with 40% bandwidth.

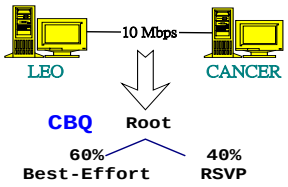
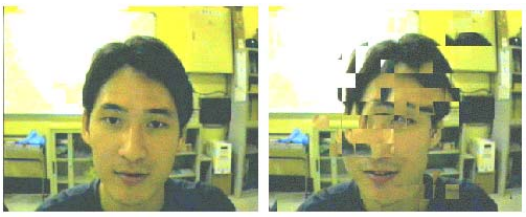


Fig. 7 Link-sharing structure

Vic with RLR

To verify the feasibility of RLR for *Vic*, a video conference is opened in *Vic* and generates about 1.2Mbps video traffic from ARIES to TAURUS and VIRGO. Background noise traffic is increasing from 3Mbps to 6Mbps. The statistics of received packets and lost packets on receivers are recorded. Fig 9 exhibits the *Vic* window observed when *Vic* is running without RLR. The *Vic* window of receiver is fragmented (see Fig 8 (b)). Fig. 9 shows that RLR has successfully made reservation for *Vic*. Table 1 gives the statistics of packet received, packet loss and loss rate.



(a) *Vic* on sender(b) *Vic* on receiver

Fig. 8 *Vic* with best-effort

(a) *Vic* on sender(b) *Vic* on receiver

Fig. 9 *Vic* with RLR

Dynamic setting of RSVP parameters

The second experiment verifies the functionality of RSVP parameters setting agent. We open a video conference in *Vic* and change the resource reserved every 10 minutes using RSVP parameters setting agent when the conference is running. Table 1 shows the statistical result. The result indicates that the more bandwidth reserved, the lower loss rate will be. This proves that the receiver can arbitrarily change its bandwidth requirement by means of the agent.

Table 1 Result of experiment 2

Background Traffic (4Mbps)				
RSVP Parameters Specified in Agent	r: token rate (KB/s)	100	150	250
	b: Bucket size (KB)	300	300	300
	p: Peak rate (KB/s)	350	350	350
Approximated BW reserved (Mbps)		0.8	1.2	2.0
Receiver Report	RTP flow volume (Kbits)	190828	345388	421547
	Packet received	27753	48416	58563
	Packet loss	2431	238	9
	Loss rate (%)	**8.05**	**0.49**	**0.02**

VI. Conclusion

In this paper, a RLR approach for UDP-based, multicast-style applications is realized. A MBone applications *Vic* is successfully transformed into RSVP-aware by means of RLR method. In addition, a RSVP parameters setting agent with a user-friendly interface is developed. It provides a convenient way for users to arbitrarily and dynamically assign their reservation requirements.

In the future, we will port RLR module for Microsoft Windows to verify the feasibility of RLR method on these platforms. Meanwhile, we will apply RLR method in the combination of IntServ and DiffServ domains.

Reference

[1] David Durham and Raj Yavatkar, "Inside the Internet's Resource reSerVation Protocol," 1999.

[2] Metz, C., "RSVP: general-purpose signaling for IP," IEEE Internet Computing, Volume: 33, Page(s): 95 -99, May-June 1999.

[3] C.K. Shieh, Y.B. Miao, J.Y. Wang, W.S. Hwang, J.F. Chiu, "A Transparent Deployment Method of RSVP-aware Applications on UNIX, accepted by IEEE-ICON, 2001.

[4] H.Schulzrinne, S.Casner, R.Frederick, V.Jacobson, "RTP: A Transport Protocol for Real-time Applications", RFC 1889, Internet Engineering Task Force, 1996

[5] M.Handley, J.Crowcroft, C.Bormann, J.Ott, "Very large conferences on the Internet: the Internet multimedia conferencing architecture", Computer Metworks Vol.31, 1999, pp191-204

[6] V. Jacobson and S. McCanne, "Vic-Video Conferencing tool", available at URL: http:// www-nrg.ee.lbl.gov/Vic/

[7] Yoram Bernet,"The Complementary Roles of RSVP and Differentiated Services," IEEE Communications Magazine, Volume:382, Feb. 2000, Page(s): 154-162.