ELSEVIER

Computer Networks 40 (2002) 45–56

COMPUTER NETWORKS

www.elsevier.com/locate/comnet

A transparent deployment method of RSVP-aware applications on UNIX

Yu-Ben Miao [a,*], Wen-Shyang Hwang [b], Ce-Kuen Shieh [a]

[a] *Department of Electrical Engineering, National Cheng Kung University, Taiwan, Taiwan, ROC*
[b] *Department of Electrical Engineering, National Kaohsiung University of Applied Sciences, Taiwan, Taiwan, ROC*

Abstract

This paper proposes a method, called RLR (RSVP (Resource reSerVation Protocol) library redirection), which can transform legacy Internet applications into RSVP-aware applications without modifying their source files by redirecting procedure calls from the socket library to the RAPI library. This method can be fulfilled for UNIX operating systems, such as Linux and Free-BSD, etc., since these operating systems support the related mechanisms of procedure call interception. In addition to the advantage of transparent transformation, RLR also allows a single RLR software module to be used for multiple programs if they have similar protocol behavior.

In this study, the RLR method has been used to render several legacy network applications RSVP aware. These applications include TCP-based and UDP-based programs, the two major Internet applications. The satisfactory use of a single RLR module for multiple applications is also verified. The results show that RLR is a feasible approach for deploying RSVP-aware applications.

Keywords: RLR; RSVP; RAPI; FTP; RTP

1. Introduction

The issue of quality of service (QoS) has been discussed extensively in the past few years. Since traditional best-effort service cannot meet the delivery requirements of time- and performance-critical applications, there has been an inevitable shift towards integration of QoS into IP networks [18], and most worldwide network equipment manufacturers, such as Cisco, Nortel and Cabletron, are now building QoS-aware routers [14–16]. Furthermore, most popular operating systems, e.g., Microsoft Windows and UNIX, now support QoS stacks in their kernels [3].

However, the shortage of QoS-aware applications has constrained a more widespread deployment of QoS networks, especially for IntServ (Integrated Service), or IntServ-over-DiffServ (Differentiated Service) networks [2–5]. In these networks, QoS-aware applications interact with QoS mechanisms to request the service quality they require [1,13,19]. However, since most legacy internet applications were developed before modern QoS mechanisms were developed, this functionality is missing in the majority of cases.

[*] Corresponding author.
E-mail addresses: ybmiau@hpds.ee.ncku.edu.tw (Y.-B. Miao), wshwang@mail.ee.kuas.edu.tw (W.-S. Hwang), shieh@eembox.ee.ncku.edu.tw (C.-K. Shieh).

Several possible approaches exist for overcoming this problem. One solution is to design QoS-aware applications from scratch. However, this is time consuming and requires a great deal of effort. Alternatively, it is possible to modify the source codes of the legacy applications to facilitate interaction with the QoS mechanism. Although this approach is likely to require less effort than the first, there are several drawbacks. Firstly, the source code must be available; secondly, it is necessary to modify each application individually, even if their protocol behaviors are identical. Furthermore, both approaches require application programmers to be aware of, and to handle, the details of the QoS mechanism. All these factors explain in part why the growth of QoS-aware applications has fallen far behind the deployment of QoS-capable devices.

This paper introduces a method, referred to henceforth as RLR (RSVP (Resource reSerVation Protocol) library redirection), which overcomes the limitations presented above. This method provides a transparent upgrade of legacy applications so that they become RSVP-aware, and requires no modification of the application source code. Furthermore, one single RLR module may be used for all applications having similar protocol behavior. To demonstrate the feasibility of the proposed RLR method, this paper considers several file transfer protocol (FTP) applications and several applications running real time protocol (RTP). The former are TCP-based applications, while the latter are UDP-based streaming applications. These applications have persistent connections during data transfer and hence require certain QoS assurance. Besides, the FTP programs used in this study have similar protocol behavior, but different user interfaces. In this way it is possible to verify the statement that "one module can be used for multiple programs". Furthermore, FTP and RTP applications both use dynamic port binding, i.e., the port number information required by the RSVP mechanism is not determined until the programs are running, and so if RLR is successful in transforming FTP and RTP applications, then it is likely that most other internet applications may also be upgraded in a similar manner.

The remainder of this paper is organized as follows: Section 2 introduces the background to the RSVP signaling protocol and RAPI interfaces, while Section 3 describes the details of the RLR method. The implementation of RSVP-aware applications using the RLR method and the experimental results are presented in Sections 4 and 5, respectively. Finally, Section 6 provides some brief concluding remarks.

2. Background

In IntServ architectures or IntServ-over-DiffServ architectures, QoS-aware applications usually employ a signaling protocol to notify the network of their resource requirements. The signaling protocol most commonly adopted is RSVP, which is basically a receiver-oriented protocol, and which requires the sender to first issue RSVP PATH messages with parameters describing the characteristics of the traffic which it is going to generate. The receiver then responds by issuing RSVP RESV messages, which include parameters specifying the resource required [7,8].

In RSVP-enabled network architectures, each RSVP host contains RSVP-aware applications, an RSVP API, an RSVP Daemon, and a Traffic Control which comprises packet scheduler and packet classifier mechanisms. RSVP-aware applications are programs which send or receive data flows and, unlike legacy applications, they interact with an RSVP daemon in order to acquire the necessary QoS provision from the network.

Each RSVP API is a set of procedures used by the applications to interact with the RSVP daemon. These procedures are generally arranged into libraries, and are linked by the programs at run time. A typical RSVP API is a RAPI library on a UNIX platform [20]. The RAPI includes the following procedures:

rapi_session(), used to initialize a session.

rapi_sender(), used to notify the RSVP daemon of the sender traffic characteristics.

rapi_reserve(), used to notify the RSVP daemon of the reservation parameters.

rapi_getfd() and rapi_dispatch(), used together to receive notification of events.

The RSVP daemon is responsible for handling RSVP signaling. It must be able to deliver RSVP messages to the network, and to process all RSVP messages received by the host. In addition, it must interact with the Traffic Control.

Within the Traffic Control, it is the responsibility of the packet classifier to identify packets corresponding to a provisioned flow, and that of the packet scheduler to ensure that packets generated by the source application are in profile with the specified QoS.

In a typical scenario, after loading the RAPI library module, an RSVP-aware application will use RAPI to initialize a QoS session and to provide a callback routine. RAPI then provides the sender's traffic characterization parameters, or the receiver's QoS specifications to the RSVP daemon. When network events relating to the initialized QoS session occur, a callback routine is invoked which notifies the application.

3. RLR

The RLR method consists of the following steps. Firstly, the relevant invocations of the socket routines from the applications are intercepted and if necessary, the parameters included in the intercepted routine calls are collected. The originally invoked socket routines continue to completion, but the control flow is redirected to call the RAPI to communicate with the RSVP daemon rather than returning directly to the applications. In response, the RSVP daemon issues appropriate RSVP signaling messages. Once these signaling procedures are completed, the control flow is returned to the applications. Other than a small time delay, the applications are unaware of any change in operations. Fig. 1 illustrates the RLR control flow.

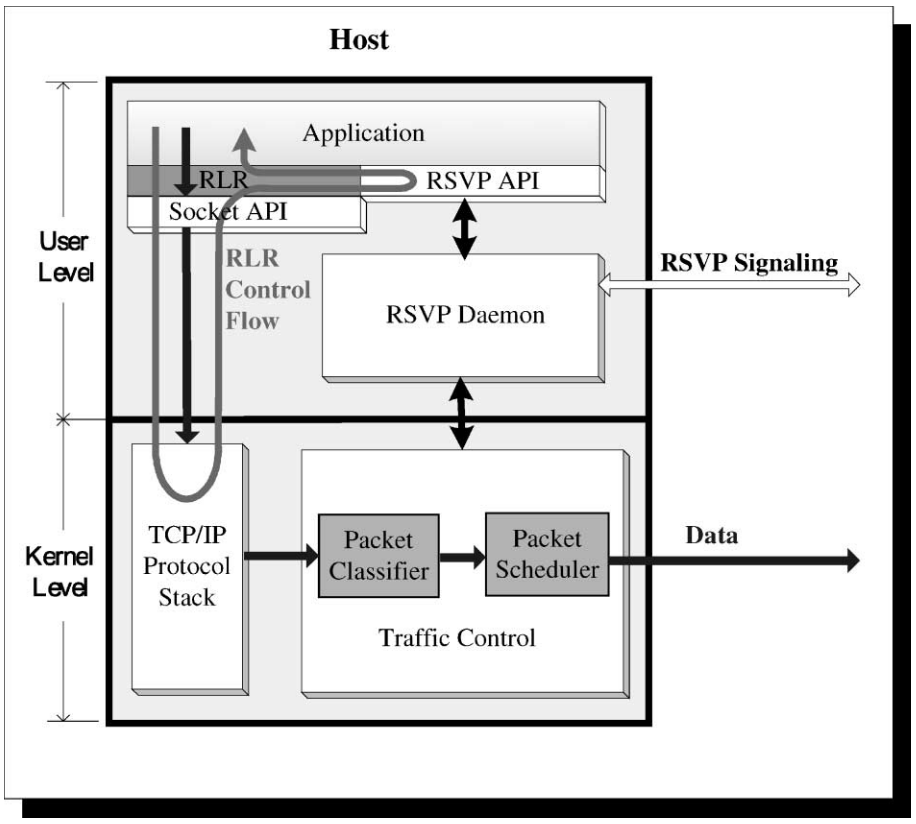


Fig. 1. Control flow of RSVP library redirection.

Consider the following example. If a client wishes to request QoS from the network for a TCP connection in order to send data to a remote server, it is necessary to intercept the *connect()* routine invoked by the client. The IP address and port number parameters, which are included in the *connect()* routine, are collected since they are part of the sender's traffic characterization parameters. The remaining parameters included in TSPEC, such as average token rate, bucket size and peak rate, can be set by the user via some kind of user interface. Having collected these parameters, the *connect()* routine then continues to completion. However, before returning to the application program, control is passed to the *rapi_sender()* routine, which notifies the RSVP daemon of the sender parameters. The RSVP daemon then sends a PATH message to the remote server.

The server must call the *accept()* routine to wait for the connection requests. Under the RLR method, when a connect request is present, control will busy-wait for the PATH message through which the flow specification can be established, before returning to the application from the *accept()* routine. On receiving the PATH message, control passes to the *rapi_reserve()* routine, which will then hand the RSVP daemon the flow specification required for responding the RESV message to the client.

The same approach can be applied to UDP-based streaming applications. In this situation, a sender transmits data packets through the UDP socket by calling the interface *sendto()* or *sendmsg()* routines, whereas the receiver retrieves a UDP packet by calling the *recvfrom()* routine. In this case, the *sendto ()* (or *sendmsg()*) and *recvfrom ()* routines need to be intercepted in order to collect the IP address and port number parameters.

The feasibility of the RLR method depends on whether or not the socket calls from the application programs can be intercepted. Fortunately, since most internet applications on current UNIX operating systems use dynamic linkage of the socket library, it is possible to prepare a RLR library in which each routine has the same name as that to be intercepted within the socket library. By using the LD_PRELOAD environment variable to replace the socket library, it is possible for the RLR library to intercept the associated socket calls [6]. The RLR library then collects the required parameters and continues the socket calls by invoking the original socket library through the *dlopen()* and *dlsym()* routines. Basically, *dlopen()* loads the shared library and maps it into memory, if it is not already loaded, while *dlsym()* retrieves the address of the symbols that are inside the library [6,12].

4. Implementation of RSVP-aware applications

In this study, the RLR method was used to render several legacy FTP applications RSVP-aware, including *ftp*, *gftp*, etc. The method was also applied to several applications running on RTP, such as *Vic* and *Rat*, which are video/audio conferencing programs having the ability to broadcast or uni-cast video/audio streams between multi-participants. Basically, *ftp*, *gftp*, etc., are TCP-based, while Vic and Rat applications are UDP-based.

4.1. FTP

FTP applications provide the function of file transfer for Internet users. Most of these applications have identical protocol behavior since a single FTP server needs to serve different FTP clients. FTP applications, such as *ftp*, *gftp*, *xftp*, *lftp*, and *ftptool* etc. use TCP connections and allow file transfer in either direction. Typically, they adopt the implementation of a concurrent server in which the server program forks a child process to serve each connection request (as shown in Fig. 2). Initially, the FTP client issues a connection request to the FTP server, which responds by forking a new child process to serve this request. As a result, a control connection is established for each client–server pair, which allows the client to transfer commands to the server. Upon receipt of a client command, the server issues a connection request to the client in order to establish a data connection. The server port numbers used for the data connection and the control connection are 20 and 21, respectively, while those on the client side are se-

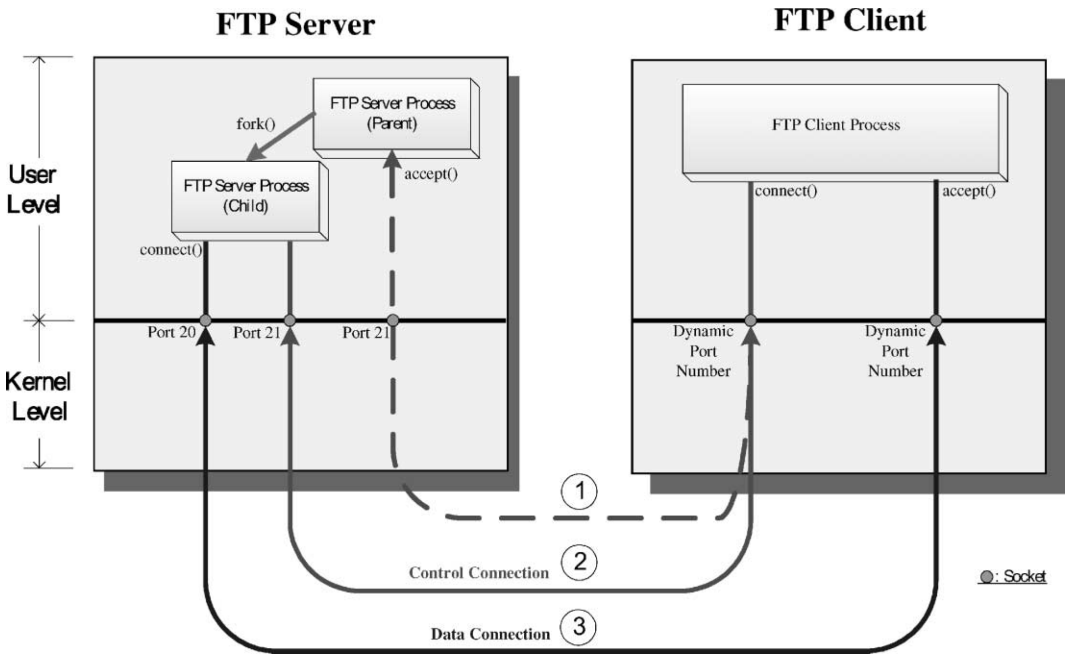


Fig. 2. Concurrent FTP server.

lected randomly (in general, they are greater than 1024).

Essentially, data and control connections for FTP applications are bi-directional, i.e., the data or commands go in one direction and the flow control packets, e.g., ACK packets, travel in the opposite direction. In order to obtain the best possible system performance, it is necessary to protect these bi-directional flows. However, since the volume of traffic through the data connection is not symmetrical, it is necessary to identify the direction in which data will be transferred, and to reserve a larger bandwidth for it if the network resources are to be used effectively.

Since the direction of data transfer depends on the commands that user inputs, it is necessary to inspect the user's commands sent in the control connection. An implication of this is that the *send()* and *receive()* routines must be intercepted, in addition to the other required function calls.

4.2. RTP

RTP is a protocol which provides end-to-end delivery services for temporally sensitive data. Its data transport is augmented by a control protocol (RTCP, RTP Control Protocol) which allows monitoring of the data delivery by providing feedback on the quality of the data distribution. Usually, both protocols run on top of UDP and use separate port numbers [17]. Vic and Rat are two typical Internet applications using RTP/RTCP.

Although both RTP applications must send and receive RTP and RTCP packets during execution, they may use different methods to do this, e.g., Vic uses *sendmsg()* to send the RTP packet, and *sendto()* to send RTCP packets. (The typical scenario is shown in Fig. 3.) By contrast, Rat transfers both RTP and RTCP traffic using just the *sendto()* routine.

In the implementation of RLR for Vic, the *connect()* and *recvfrom()* routines are intercepted in order to collect the IP addresses and port numbers required for RSVP reservation. However, in the case of the Rat application, the IP addresses and port numbers are not determined until the invoking of *sendto()* routine, and so it is necessary to intercept the *sendto()* and *recvfrom()* functions instead.

Since applications running RTP/RTCP invoke the UDP socket interface for their packet transfer,

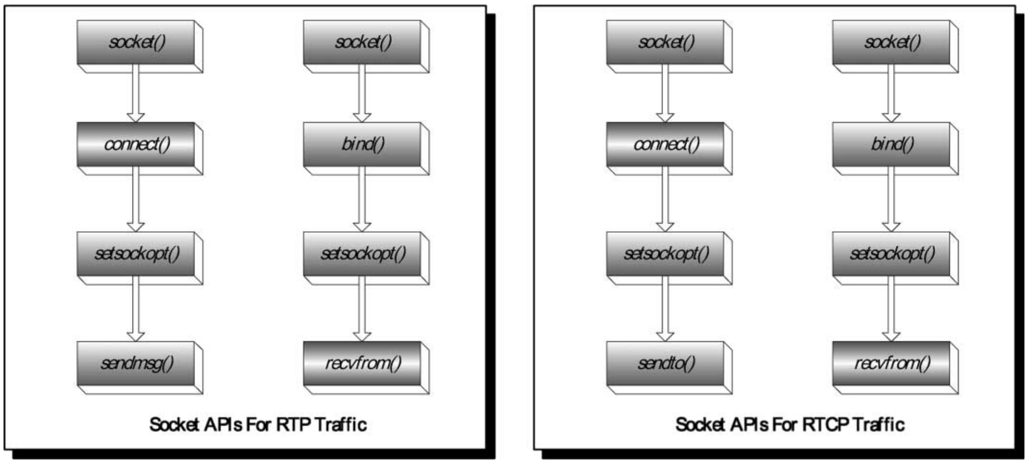


Fig. 3. Socket APIs invoked in Vic.

the RLR method can be applied to provide them with RSVP protection. However, unlike the implementation of RSVP-aware FTP applications, it is sufficient to make bandwidth reservation in one direction only since UDP is uni-directional.

5. Experimental results

Four experiments are conducted in order to evaluate the feasibility of the RLR method. The first experiment verifies the effectiveness of the RLR method, while the second experiment measures its overhead. The aim of the third experiment is to determine whether a single RLR module can be used successfully for more than one FTP program, and the final experiment considers the influence of reservation on TCP flow control.

Fig. 4 shows the experimental environment used to conduct these experiments. It will be seen that there are 4 hosts, and 2 Cisco 2621 routers, which are RSVP enabled. The routers are configured so as to form an IntServ domain. 10 Mbps ethernet links are used throughout. Weighted fair queue (WFQ) is implemented on routers and provides traffic priority management that automatically sorts among individual traffic streams. Each of the hosts is equipped with a P-II 266 CPU having 64M of RAM. Hosts A and B are both FreeBSD3.2 platforms. Host A is used as the FTP server and the RTP sender, and Host B acts as the FTP client and the RTP receiver. Both host machines are installed with class based queue (CBQ), and the RSVP daemon. CBQ is a traffic controller, which manages the resources of a link based upon arbitrarily defined traffic classes [9]. In the experiments, it carries out the functions of the Traffic Control, and cooperates with the RSVP daemon to ensure that the RSVP messages signal properly.

Hosts C and D emulate background traffic by using Mgen to generate best-effort UDP packets. Mgen also provides the statistical ability to measure the performance of an IP network [10]. The traffic is logged by tele traffic tapper (TTT). TTT is a graphical tool which provides real time traffic monitoring [11].

5.1. Effectiveness of RLR method

5.1.1. RLR with TCP

To verify the effectiveness of RLR for TCP-based applications, a 20 Mbyte file is retrieved from the FTP server (Host A) by the FTP client; firstly by an RSVP-unaware (legacy) application, and then by an application rendered RSVP-aware by the RLR method. While this retrieval process is underway, background traffic is generated, and transferred from Host C to Host D. The volume of background traffic is varied, and the variation in throughput between Host A and Host B is measured. In order to observe the effectiveness of RLR

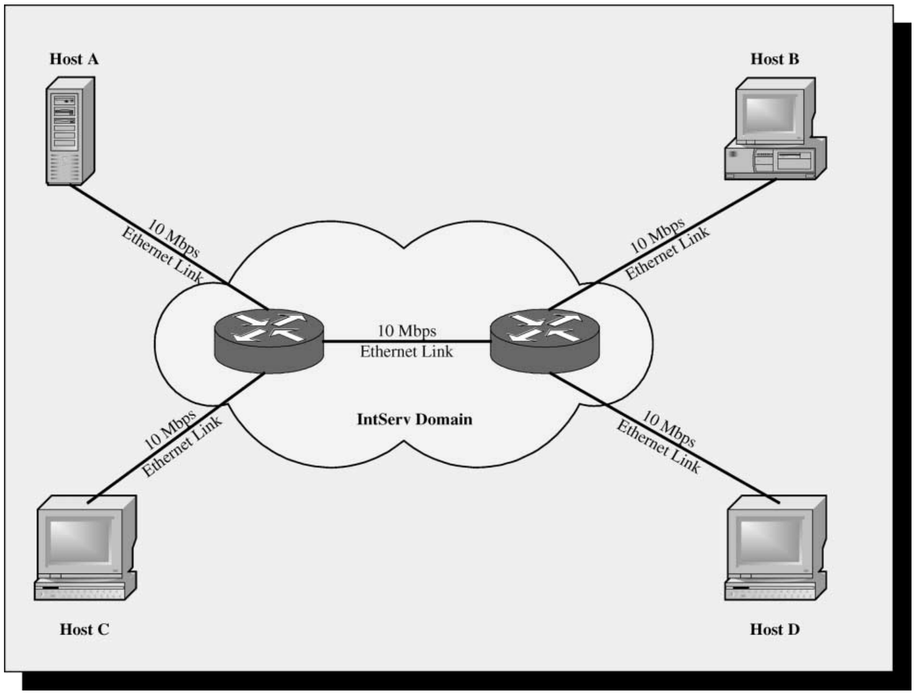


Fig. 4. Experimental environment.

manifestly, the throughput of RSVP-aware FTP applications is not measured until the successful establishment of RSVP reservation. The WFQ on routers allocate the RSVP traffic 40% of the bandwidth, and the best-effort traffic, 60%.

Figs. 5 and 6 are traffic volume recorded by TTT at an interval of 1 s when FTPs are transmitting files. Fig. 5 shows the throughput under the legacy FTP application, for different background traffic loads from 0 to 6 Mbps. It will be seen that the throughput is indeed influenced by the background traffic, and that the throughput is almost zero in some cases. Fig. 6 shows the same circumstances for an RSVP-aware, FTP-based application. Since the throughput remains stable as the volume of background traffic is increased, it is clear that the RLR method has successfully established an RSVP protected connection.

5.1.2. RLR with UDP

To verify the effectiveness of the RLR method for UDP-based applications, a video stream is unicast from the RTP sender (Host A) to the RTP receiver (Host B); firstly by an RSVP-unaware (legacy) RTP application, and then by an RTP application rendered RSVP-aware by the RLR method. Simultaneously, background traffic is generated, and transferred from Host C to Host D. The volume of background traffic is varied, and the corresponding variation of QoS is measured for both RTP applications. The WFQ on routers is configured as before, i.e., 40% for RSVP traffic, and 60% for best-effort traffic.

Table 1 shows the quality of service for Vic for background traffic volumes of 3 and 6 Mbps. Fig. 7 shows the throughput of legacy and RSVP-aware Vic, for different background traffic loads from 0 to 6 Mbps. It can be seen that there is significant packet loss under heavy network loads for legacy Vic, but that when Vic is rendered RSVP-aware, an RSVP reserved connection is successfully established, which provides a better quality of service. A similar outcome is obtained for the Rat application Fig. 8. A comparison of the Table 1 and 2 shows that Rat suffers more severe packet loss than Vic when both applications are RSVP-unaware. This is to be expected since audio data is more temporal-sensitive than video data.

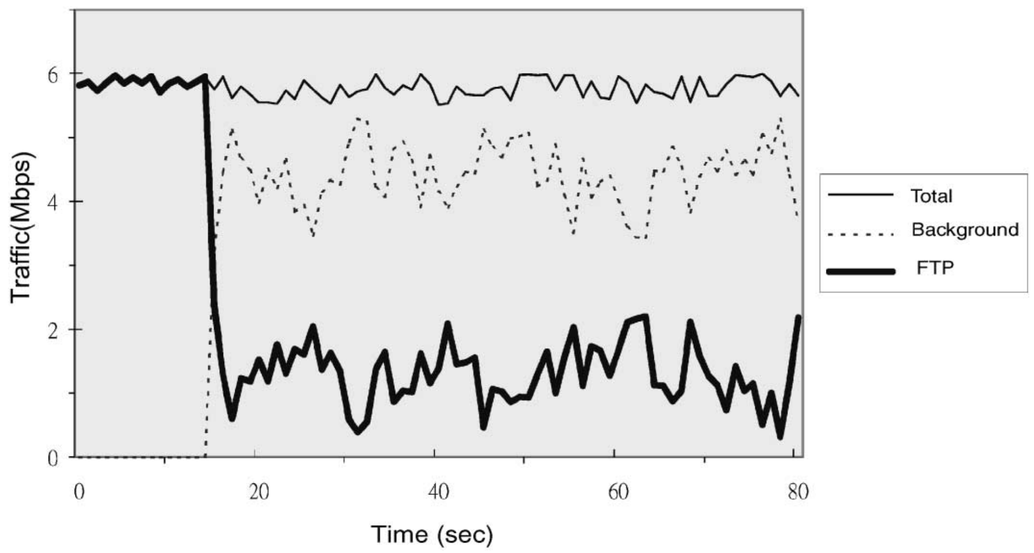


Fig. 5. Throughput of RSVP-unaware (legacy) FTP.

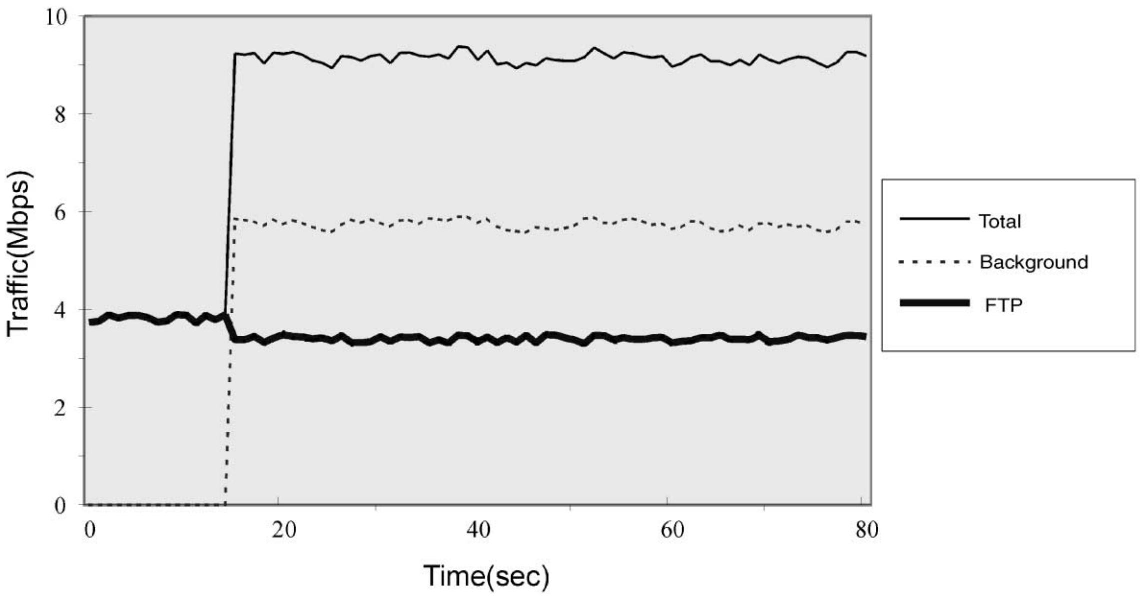


Fig. 6. Throughput of RSVP-aware (using RLR method) FTP.

5.2. Overhead of RLR

The interception and re-direction of appropriate socket routine calls issued by an application (e.g., *connect( ), accept( ), close( ), send( ), receive( ), sendto( ),* and *recvfrom( )*, etc.), which is the essential part of the RLR method, induces an overhead. The interception of the first three socket

Table 1
Statistical results of QoS on Vic

Application	RSVP-unaware (legacy) Vic		RSVP-aware (using RLR method) Vic	
Background traffic (MB)	3	6	3	6
Packet received	47512	45080	58081	56787
Packet loss	37	12327	540	604
Loss rate(%)	0.08	21.47	0.92	1.05

routines influences the time required to set up and tear down a connection, while the remaining routines cause a processing overhead for data transfer.

In order to evaluate the overhead of RLR for TCP-based applications, the performance of two types of RSVP-aware FTP applications is measured. In one application the RAPI invocations are coded directly within the source code (i.e., an embedded approach), while in the other, the RLR method is adopted. The same program is used for the two FTP applications.

To compare the time for setting up and tearing down a connection between the FTP client and the server, the average time required to set up a connection and then immediately tear it down is measured for the two FTP applications. The results are presented in Table 3.

The average time required to set up, and to tear down a connection using the RLR method is 1.7 times higher than when using the embedded approach. Hence RLR is not appropriate for non-persistent connections such as WWW applications, etc. Nonetheless, for FTP, this overhead can be neglected since the connection activity occurs only once during a FTP session.

To evaluate the overhead on the processing for data transfer, the average time to transfer 15 K, 150 K, 1 M, and 1.5 M is measured in the same environment as described above (in Section 5.1.1). The results are shown in Table 4.

Similar results were observed for UDP-based streaming applications. Basically, the overhead of setting up an RSVP session occurs only once at the

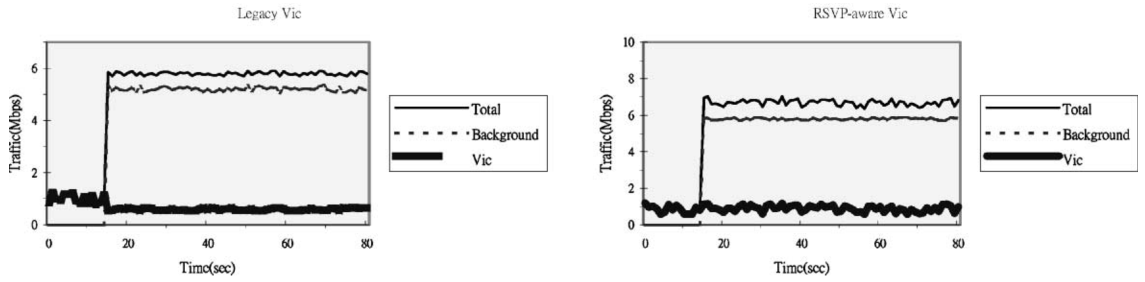


Fig. 7. Throughput of Vic.

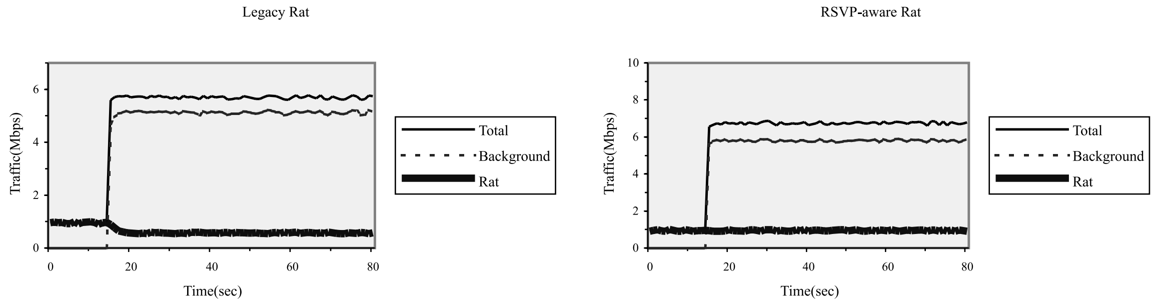


Fig. 8. Throughput of Rat.

Table 2
Statistical results of QoS on Rat

Application	RSVP-unaware (legacy) Rat		RSVP-aware (using RLR method) Rat	
Background traffic (MB)	3	6	3	6
Packet received	114704	34427	114704	111175
Packet loss	0	69182	0	3
Loss rate(%)	0.00	66.77	0.00	0.00

Table 3
Average time to set up and tear down a connection

Embedded approach elapsed time (ms)	RLR approach elapsed time (ms)
3.790	6.409

Table 4
Data transfer time for embedded and RLR approach

File size (bytes)	Embedded approach elapsed time (ms)	RLR approach elapsed time (ms)
15K	1604	1620
150K	7161	7182
1M	41729	42001
1.5M	62527	62593

first invocation of *sendto()* routine in these cases, and the processing overhead is comparable to those of the RSVP-aware FTP applications.

The experimental results demonstrate that the RLR method induces less than 1% overhead in this particular environment. However, the degree of overhead depends on the processing power of the hosts, and since the speed of commodity CPUs is increasing very rapidly, this overhead will progressively diminish.

5.3. A single RLR module for multiple programs

To verify whether or not a single RLR module can work successfully for multiple FTP applications, several versions of FTP applications are tested separately using a single RLR module. These applications include ftp, gftp, xftp, lftp, and ftptool. Although these applications have different user interfaces, they all link to their socket library dynamically and have similar protocol behaviors. Therefore, they are suitable for use in confirming the achievability of “a single RLR module for multiple programs”. The environment setting is the same as described in Section 5.1.1. Each application runs on Host B and retrieves a 10M bytes file from Host A. A same RLR module is used throughout. Statistics relating to throughputs are presented in Table 5.

Table 5
A single RLR module for multiple FTP applications

Application name	RSVP-unaware application throughput (Mbps)	RSVP-aware application throughput (Mbps)
ftp	1.05	3.77
gftp	0.98	3.57
xftp	0.92	3.42
lftp	1.02	3.63
ftptool	0.93	3.20

The results show that RLR has successfully set up an RSVP protected connection for these FTP applications and confirm that a single RLR module can transform several legacy FTP applications into RSVP-aware applications.

5.4. The influence of reservation on TCP flow control

It is well known that TCP is a connection-oriented protocol, i.e., it relies on traffic feedback to handle flow control and error recovery. Each end host of a TCP connection returns an ACK packet whenever it receives a data packet.

To evaluate the influence of reservation on TCP flow control, two different RLR modules are designed. One module provides uni-directional reservation for non-symmetric TCP traffic such as FTP, while the other provides bi-directional reservation. The only difference between these two types of reservation is that the ACK traffic in the bi-directional reservation is protected, whereas that in the uni-directional reservation is not.

In this experiment, background traffic is generated from Host D to Host C in order that it competes with the ACK packets for bandwidth.

Table 6
Influence of reservation on TCP flow control

Back-ground traffic (Mbps)	TCP throughput with bi-directional reservation (Mbps)	TCP throughput with uni-directional reservation (Mbps)
2	3.55	3.56
4	3.57	3.55
6	3.52	3.23

The setting of reservation for data flow is the same as Section 5.1, i.e., 40% for RSVP traffic, and 60% for best-effort traffic. In the opposite direction, 60% bandwidth is allocated for the best-effort traffic, and a 5% band width is reserved for the ACK packets in the bi-directional reservation.

Observation of Table 6 shows that ACK packets are delayed or dropped under heavy load conditions. It should be noted that the failure to return flow control packets has a significant influence on the throughput of data traffic.

6. Conclusions

This paper has proposed a RLR method, which can transform legacy Internet applications into RSVP-aware applications without modification of their source files. Several TCP-based and UDP-based applications have been successfully rendered RSVP-aware using this method. The ability of a single RLR module to transform multiple legacy applications into RSVP-aware applications has also been proven. Although the RLR method does induce some overhead, the measured operating costs still remain within a reasonable range. The influence of reservation on TCP flow control has also been investigated. The experiments have shown the feasibility of using the RLR method on UNIX platforms.

In the future, it is the current authors' intention to use the RLR method to render multicast applications RSVP-aware. Meanwhile, a RLR module will be constructed for Microsoft Windows to verify the feasibility of the RLR method on these platforms.

References

[1] P.G.S. Florissi, Y. Yemini, D. Florissi, QoSockets: A new extension to the sockets API for end-to-end application QoS management, Computer Networks 35 (2001) 57–76.

[2] C. Metz, RSVP: General-purpose signaling for IP, IEEE Internet Computing 33 (1999) 95–99.

[3] Y. Bernet, The complementary roles of RSVP and differentiated services, IEEE Communications Magazine 382 (2000) 154–162.

[4] A. Detti, M. Listanti, L. Veltri, Supporting RSVP in a differentiated service domain—an architectural framework and a scalability analysis, in: IEEE International Conference on Communications, vol. 1, 1999, pp. 204–210.

[5] J. Schmitt, M. Karsten, L. Wolf, R. Steinmetz, Aggregation of guaranteed service flows, in: Seventh International Workshop on Quality of Service, 1999, pp. 147–155.

[6] D.A. Curry, UNIX systems programming for SVR4, first ed., 1996.

[7] L. Zhang, S. Deering, D. Estrin, S. Shenker, D. Zappala, RSVP: A new resource ReSerVation protocol, IEEE Network 7 (5) (1993) 8–18.

[8] D. Durham, R. Yavatkar, in: Inside the Internet's Resource reSerVation Protocol, Wiley, New York, 1999, pp. 127–192.

[9] S. Floyd, M.F. Speer, Experimental results for class-based queuing, http://www-nrg.ee.lbl.gov/floyd/cbq/notes.html 1998.

[10] Brian Adamson, The MGEN Toolset, http://manimac.itd.nrl.navy.mil/MGEN, 1997.

[11] Kenjiro Cho, A public release of ALTQ for FreeBSD http://www.csl.sony.co.jp/person/kjc/software.html, 2000.

[12] W. Richard Stevens, UNIX Network Programming 1, second ed., 1998.

[13] P.Y. Wang, Y. Yemini, D. Florissi, J. Zinky, P. Florissi, Experimental QoS performances of multimedia applications, INFOCOM, vol. 2, 2000, pp. 970–979.

[14] E. Basturk, A. Birman, G. Delp, R. Guerin, R. Haas, S. Kamat, D. Kandlur, P. Pan, D. Pendarakis, V. Peris, R. Rajan, D. Saha, D. Williams, Design and implementation of a QoS capable switch-router, Computer Networks 31 (1999) 19–32.

[15] http://www.cisco.com.

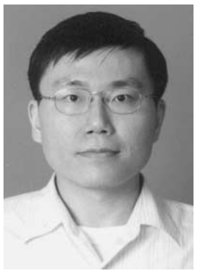
[16] http://www.nortelnetworks.com/index.html.

[17] J. Rosenberg, Sampling of the group membership in RTP, RFC 2762, 2000.

[18] D.T. McWherter, J. Sevy, W.C. Regli, Building an IP network quality-of-service testbed, IEEE Internet Computing 44 (2000) 65–73.

[19] S.N. Bhatti, G. Knight, Enabling QoS adaptation decisions for internet applications, Computer Networks 31 (1999) 669–692.

[20] R. Braden, D. Hoffman, RAPI-an RSVP application programming interface version 5, Internet Draft, August 11, 1998.



Yu-Ben Miao is currently a Ph.D. candidate studying at Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan. Miao received his BS degree from National Chao Tung University in 1995 and MS degree from National Cheng Kung University in 1997. His current research interests include QoS, RSVP, wireless network and WWW database applications.

Wen-Shyang Hwang received the B.S., M.S., and Ph.D. degrees in Electrical Engineering from National Cheng-Kung University, Taiwan, in 1984, 1990 and 1996, respectively. He is an associate professor of Electrical Engineering, National Kaohsiung University of Applied Sciences, Taiwan. His current research interests include multichannel WDM networks, performance evaluation, QoS, RSVP, WWW database applications.

Ce-Kuen Shieh is currently a professor teaching at the Department of Electrical Engineering, National Cheng Kung University. He received his Ph.D., MS, and BS degrees from Electrical Engineering Department of National Cheng Kung University, Tainan, Taiwan. His current research interests include distributed and parallel processing systems, computer networking, and operating systems.