Proceedings

Ninth IEEE International Conference on Networks

October 10 – 12, 2001
Bangkok, Thailand

Sponsored by
IEEE Singapore Chapter
IEEE Thailand Chapter
Mahanakorn University of Technology

IEEE COMPUTER SOCIETY

Los Alamitos, California

Washington • Brussels • Tokyo

13. ATM Networks

14. Mobile and Wireless Networks

15. Quality of Service

A Transparent Deployment Method of RSVP-aware Applications on UNIX

Ce-Kuen Shieh
Department of Electrical Engineering, National Cheng Kung University, Taiwan R.O.C.
shieh@eembox.ee.ncku.edu.tw

Yu-Ben Miao
Department of Electrical Engineering, National Cheng Kung University, Taiwan R.O.C.
ybmiau@hpds.ee.ncku.edu.tw

Chen-Yu Wang
Department of Electrical Engineering, National Cheng Kung University, Taiwan R.O.C.

Wen-Shyang Hwang
Department of Electrical Engineering, National Kaohsiung University of Applied Sciences, Taiwan R.O.C.

Ji-Feng Chiu
Department of Electrical Engineering, National Kaohsiung University of Applied Sciences, Taiwan R.O.C.

Abstract

This paper proposed a method, called RLR (RSVP Library Redirection), which can transform legacy Internet applications into RSVP-aware applications without modifying their source files. RLR achieves the transparent transformation by redirecting procedure calls from the socket library to the RAPI library. For UNIX operating systems, such as Linux and Free-BSD etc., the redirection can be realized since the related mechanisms of procedure call interception are supported in these operating systems. In addition to the advantage of transparent transformation, RLR can allow single RLR software module to be used for multiple programs if they have similar behavior.

We have transformed several FTP applications to be RSVP-aware by using one RLR module. The experimental results show the feasibility of RLR.

1. Introduction

Quality of Service (QoS) has being intensively discussed in past few years. Traditional best-effort service cannot meet the delivery needs of time- and performance-critical applications. Hence, integrating QoS into IP networks seems inevitable [1]. Currently, most worldwide network manufactures, such as Cisco, Nortel and Cabletron, are fabricating QoS-aware routers [15][16]. Furthermore, most popular operating systems, like Microsoft windows and UNIX, have supported QoS stacks in their kernels [3].

But, the shortage of QoS-aware applications prevents the further deployment of QoS networks, especially for Integrated Service (IS) or Integrated Service plus Differentiated Service (DS) networks [2][3][4][5]. In these networks, QoS-aware applications must interactive with QoS mechanisms in order to request proper service quality from the network [2][3]. However, most legacy Internet applications were developed before modern QoS mechanisms being proposed. They are unaware of any QoS mechanisms.

Several possible approaches can be used to solve this problem. One is to redesign QoS aware applications from scratch. However, this approach is time consuming and costs a lot of effort. Alternatively, we can modify the source of legacy applications to add codes that directly interact with underlying QoS mechanisms. Although, this may save some effort compared with above approach, the source codes must be available. Moreover, each of applications needs to be modified separately, even if their behaviors are identical. Consequently, the growth of QoS-aware applications falls far behind the deployment of QoS-capable devices.

This paper proposed a method, named RLR (RSVP, Resource reSerVation Protocol, Library Redirection), to overcome this problem. This method can transparently upgrade legacy applications to be RSVP-aware without any modification of source code. Furthermore, for the applications with the similar behavior, one RLR module can be used for all of them. In this paper, FTP (File Transfer Protocol) applications are used to show the feasibility of the RLR method. We choose FTP applications since there are several different FTP programs in our test bed. These programs have the similar behavior but different user interfaces. So, they can be used to verify the functionality of "one module can be used for multiple programs". On the other hand, FTP applications use dynamic port binding. If RLR succeeds in this challenge then most of other Internet applications can be applied in a similar manner.

1531-2216/01 $10.00 © 2001 IEEE

The rest of this paper organizes as follows: Section 2 introduces the background about the RSVP signaling protocol and RAPI interfaces. The RLR method is described in Section 3. Section 4 illustrates the implementation of RSVP-aware FTP applications by using RLR. The experimental results are shown in Section 5. Finally, Section 6 gives a conclusion remark.

2. Background

In IS or IS+DS network architectures, a signaling protocol is usually required by QoS-aware applications to notify the network their requirements of resources. The most common signaling protocol adopted by these QoS architectures is RSVP. Basically, RSVP is a receiver-initiated protocol. The sending node is just to pass the requirements of the traffic to the receiver via sending a PATH message. The receiving node is responsible for initiating the resource reservation by sending back a RESV message.

In RSVP-enabled network architectures, each RSVP host will contain RSVP-aware applications, an RSVP API, a RSVP Daemon, and a RSVP protocol stack that consists of admission control, packet scheduler and packet classifier. RSVP-aware applications are programs that send or receive data flows. Unlike legacy applications, they must interact with RSVP daemon to require the QoS support from network.

Each of RSVP APIs is a set of procedures used by applications to interact with RSVP daemon. In general, they are created into libraries and linked by programs at run time. Typical RSVP API is RAPI library on UNIX platforms [14]. The RAPI procedures consist of *rapi_session()*, that is used to initialize a session, *rapi_sender()*, to notify RSVP daemon of the sender traffic characteristics, *rapi_reserve()*, to notify RSVP daemon of the reservation parameters, and *rapi_getfd()* as well as *rapi_dispatch()*, those are together used to receive notification of events.

The RSVP daemon is responsible for handling the RSVP signaling. It must be able to deliver RSVP messages to the network and all RSVP messages received by this host must be passed to it. Besides, the RSVP daemon has to interact with the RSVP protocol stack.

In the RSVP protocol stack, admission control is responsible for keeping track of resource consumption on a particular interface and making sure sufficient resources are available to support a new resource request. Packet classifier is responsible for identifying packets corresponding to a provisioned flow. The packet scheduler is responsible for ensuring 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 provide a callback routine. It then provides sender's traffic characterization parameters or receiver's QoS specification to the RSVP daemon through RAPI. When network events occur with respect to the initialized QoS session, a callback routine will be invoked to notify the application.

3. RLR

RLR method consists of the following steps. 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, which will issue associated RSVP signaling messages. Fifth, after finishing the signaling procedures, the control flow is returned to the applications. In this case, the applications do not sense any change of operations except a little time delay. Figure 1 shows the control flow of RLR.

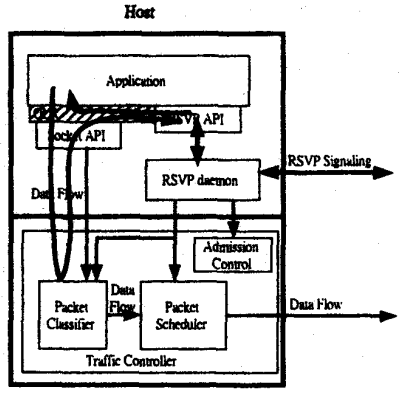

Fig 1 The Control Flow of RSVP Library Redirection

For example, if a client wants to request the QoS from the network for a TCP connection in order to send data to a remote server, the *connect()* routine invoked by the client need to be intercepted. The parameters of IP address and port number included in the *connect()* are collected since they are part of the sender's traffic characterization parameters, TSPEC (the rest of the parameters of the TSPEC, e.g. average token rate, bucket size etc, can be set by the user through some kinds of user interface). The *connect()* routine continues to complete. Before returning to the application program from *connect()* routine, control passes to *rapi_sender()* which will notify the RSVP daemon of the sender parameters. Finally, the RSVP daemon will send a PATH message to the remote server.

On the server side, a server must call *accept()* to wait the next connection request. In the RLR method, when a

connect request is present, before returning to the application from *accept()*, control will busy-wait for the PATH message through which the flow specification can be setup. On receiving the PATH message, control passes to the *rapi_reserve()* routine which will hand the RSVP daemon the flow specification for sending back the RESV message.

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

4. Implementation of RSVP-aware FTP applications

In this section, we will describe the implementation of RSVP-aware FTP applications by using RLR method. Most FTP applications use TCP connections and allow file transfer in either direction. Typically, they adopt the implementation of concurrent server in which the server program forks a child process to serve each connection request (as shown in Figure 2). For each pair of client and server, one control connection is created for the client to transfer commands to the server. On receipt of a command, the server will issue a connect request to the client to establish a connection for data transfer. The port numbers for control connection and data connection on the server side are 21 and 20, respectively, while those on the client side are randomly selected (in general, they are great than1024).

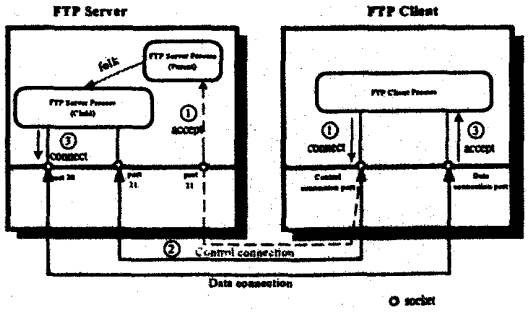

Fig. 2: Concurrent FTP Server Model

Since data and control connections for FTP applications are created through the invocations of socket interface. RLR method can be used to provide the RSVP protection for each of them. Basically, either data or control connections are bi-directional; the data or commands go in one direction and ACK packets travel in the opposite direction, it is necessary to make bi-directional protections for them in order to obtain better performance. However, the traffic volume of the data connection is not symmetric; the traffic in the direction for sending data is larger than that for sending ACK packets. That means we have to identify which direction is for data transfer and reserve a larger bandwidth for it.

In FTP applications, the direction of file transfer depends on the command a user inputs from a client node. In our implementation of RSVP-aware FTP applications, we identify the direction of file transfer by examining the user's commands transferred in the control connection. In this case, the *send()* and *receive()* need to be intercepted for tracking user commands.

5. Experimental Results

To estimate the feasibility of RLR method, we carry out 3 experiments. First experiment is to verify the effectiveness of RLR method. The overhead of RLR method is measured in the second experiment. Last one is to check whether a single RLR module can work for multiple FTP programs,

Figure 3 shows the testing environment for our experiment, in which there are 4 hosts and 2 routers. Router 1 and 2 are Cisco 2600 routers, which are RSVP enabled. Their interfaces are 10 Mbps Ethernet links.

Host A and B are FreeBSD3.2 platforms, the former is used as the FTP server and the later is the FTP client. Both of them are installed with CBQ (Class Based Queue) and RSVP daemon. CBQ is a traffic controller that can manage resources on a link based on arbitrarily defined traffic classes. It can be configured to communicate with RSVP daemon.

Host C and D emulate different background traffic by using Mgen to generate best-effort UDP packets. Mgen provides the statistics ability to measure the performance of IP network [10]. We use TTT (Tele Traffic Tapper) as a traffic monitor, which is a graphical tool that can provide real time traffic monitoring [11].

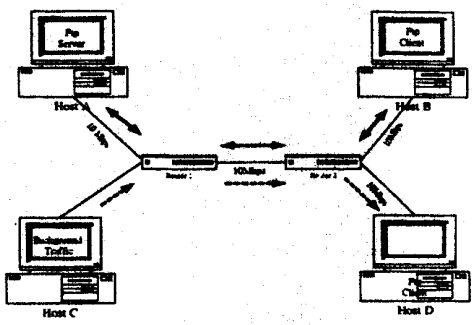

Fig. 3 Experimental Environment

5.1 Effectiveness of RLR method

To verify the effectiveness of RLR, we let RSVP-unaware (legacy) and RSVP-aware (using RLR method) FTP applications separately retrieve a 20 Mbytes file from the FTP server (host A) to the FTP client (host B). At the same time, the background traffic is generated and transferred from host C to host D. Then we vary the background traffic and measure the variance of throughput for either FTP applications. The CBQ on host A and B are configured as 2 classes: 80% for RSVP traffic and 20% for best-effort traffic.

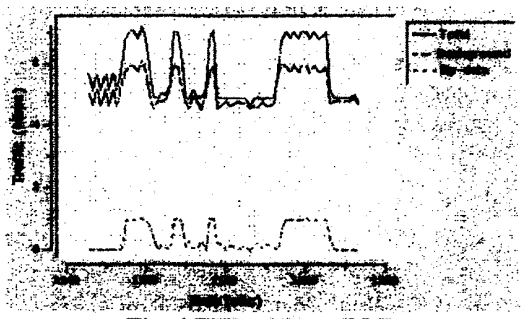

Fig. 4 FTP without RLR

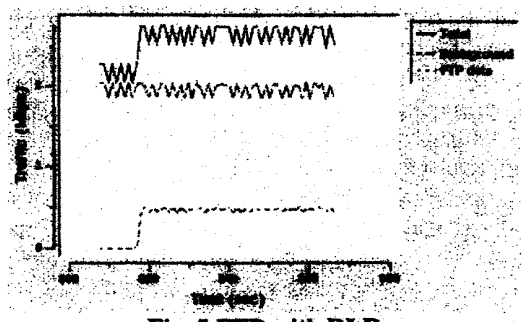

Fig. 5 FTP with RLR

Figure 4 shows that the traffic of the legacy FTP is indeed influenced by the background traffic. The throughput can even down to near zero. In figure 5, we can see RLR method has successfully set up a RSVP protected connection for RSVP-aware FTP. The throughput is stable while changing the amount of background traffic.

5.2 Overhead of RLR

In the RLR method, the related socket routine calls (e.g. connect(), accept(), close(), send(), receive(),etc.) issued by an application have to be intercepted for redirection. This will somewhat induce overhead. The interception of the former three socket routines will influence the time to setup and tear down a connection, while that of the latter two routines will cause the overhead on the processing for data transfer.

To evaluate these overheads of RLR, we measure the performance for two kinds of RSVP-aware FTP applications; one is directly coded with RAPI invocations in the source codes (embedded approach), the other is to use RLR method. We use the same program for these two FTP applications.

We measure the average time to setup and tear down a connection between two directly connected FTP client and server for these two kinds of FTP programs. The results are shown in Table.1.

Table 1 Average Time to Setup and Tear down a Connection (msec)

Embedded approach	RLR approach
3.790	6.409

Although RLR approach spends a 1.7 times average time to setup and tear down a connection compared with the embedded approach, the overhead can be neglected since the connection activity only happens once during a FTP session,

To evaluate the overhead on the processing for data transfer, we measure the average time to transfer 64K Bytes on the same environment as the above. The results are shown in Table.2.

Table 2 Data Transfer Time (msec)

Data Size	Embedded approach	RLR approach
64K Bytes	421.298	434.342

Our experiment shows that RLR method will induce around 3% overhead in this case. However, this overhead depends on the processing power of hosts. In fact, the speed of commodity CPUs increases very rapidly, this overhead will getting smaller.

5.3 One RLR module for multiple programs

To verify whether a single RLR module can work for multiple FTP applications, we have separately tested several versions of FTP applications by using a single RLR module. These applications include *ftp*, *gftp*, *xftp*, *lftp*, and *ftptool*. All of these FTP applications link socket library dynamically and have similar behavior but with different user interfaces. The result shows that one RLR module can successfully transform these legacy FTP applications into RSVP-aware applications.

6. Conclusion

This paper proposed a RLR method that can transform legacy Internet applications into RSVP-aware applications without modification of their source files. We have successfully transformed several FTP applications to be RSVP-aware by using a single RLR module. The experiments have shown the feasibility of RLR method on the UNIX platforms.

In the future, we are going to transform the UDP-based applications with or without the multicasting function to be RSVP-aware by using the RLR method. Meanwhile, we will construct a RLR module for MS Windows to verify the feasibility of RLR method on these platforms.

Reference

[1] McWherter, D. T.; Sevy, J.; Regli, W.C. "Building an IP network quality-of-service testbed," *IEEE Internet Computing*, Volume:44, July-Aug. 2000, age(s): 65-73

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

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

[4] Detti, A.; Listanti, M.; Veltri, L.,"Supporting RSVP in a Differentiated Service Domain- An Architectural Framework and a Scalability Analysis", *IEEE International Conference on Communications 1999*, Page(s): 204-210 vol. 1.

[5] Schmitt, J.; Karsten, M.; Wolf, L; Steinmetz, R.,"Aggregation of Guaranteed Service Flows", *Seventh International Workshop on Quality of Service 1999*, Page(s): 147-155

[6] David A. Curry, "*UNIX Systems Programming for SVR4*", 1st Edition July 1996.

[7] Zhang, L.; Deering, S.; Estrin, D.; Shenker, S.; Zappala, D., "RSVP: a new resource ReSerVation Protocol", *IEEE Network* Volume: 7 5, Page(s): 8 -18, Sept. 1993.

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

[9] Sally Floyd and Michael Francis Speer, "Experimental Results for Class-based Queuing", http://www-nrg.ee.lbl.gov/floyd/cbq/notes.html, Jan. 1998.

[10] Brian Adamson, "The MGEN Toolset", http://manimac.itd.nrl.navy.mil/MGEN, July 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*" Vol. 1, 2nd Edition, 1998

[13] Wang, P.Y.; Yemini, Y.; Florissi, D.; Zinky, J.; Florissi, P., "Experimental QoS performances of multimedia applications", *INFOCOM 2000*, Volume: 2, 2000, Page(s): 970 - 979 vol.2.

[14] R. Braden and D. Hoffman, "RAPI-An RSVP Application Programming Interface version 5", *Internet Draft*, August 11, 1998.

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

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