On the introduction of quality of service awareness in legacy distributed applications

Article · July 2002

DOI: 10.1145/568760.568874

5 authors, including:

Roberto Canonico
University of Naples Federico II
66 PUBLICATIONS **405** CITATIONS
SEE PROFILE

Maurizio D'Arienzo
Second University of Naples
41 PUBLICATIONS **205** CITATIONS
SEE PROFILE

Simon Pietro Romano
University of Naples Federico II
104 PUBLICATIONS **478** CITATIONS
SEE PROFILE

Giorgio Ventre
University of Naples Federico II
56 PUBLICATIONS **286** CITATIONS
SEE PROFILE

On the introduction of Quality of Service awareness in legacy distributed applications

R. Canonico*, M. D'Arienzo*, B. Fadini*, S.P. Romano* and G. Ventre*°

*Dipartimento di Informatica e Sistemistica
Università di Napoli "Federico II"
Via Claudio 21 – 80125 - Napoli - ITALY
{rcanonico, maudarie, fadini, spromano}@unina.it

° Laboratorio Nazionale ITEM
Consorzio Interuniversitario Nazionale per l'Informatica
Via Diocleziano 328 – 80126 – Napoli – ITALY
giorgio.ventre@napoli.consorzio-cini.it

ABSTRACT

A number of distributed applications require communication services with Quality of Service (QoS) guarantees. Work undertaken within the Internet Engineering Task Force (IETF) has led to the definition of novel architectural models for the Internet with QoS support. According to these models, the network has to be appropriately configured in order to provide applications with the needed performance guarantees. In a first proposal, called Integrated Services, applications need to explicitly interact with network routers by means of a signaling protocol (such as RSVP), in order to enforce QoS on a per-flow basis. The Differentiated Services architecture, on the other hand, looks after scalability, thus providing performance guarantees to aggregates of flows. In the case of real-time applications, a hybrid model capable of putting together micro-flow guarantees in the access network and aggregate management in the backbone seems to represent the ideal tradeoff between strict performance and scalability. In this scenario, giving applications a means to interact with the underlying QoS services is of primary importance. Hence, several special-purpose APIs have been defined to let applications negotiate QoS parameters across QoS-capable networks. However, so far, none of these APIs is available for the use of programmers in different operating environments. We believe that such features should be embedded in programming environments for distributed applications. In this work we present how we included QoS control features in a programming language that since years has been adopted for the development of network-based applications: Tcl. We present QTcl, an extension of Tcl, which provides programmers with a new set of primitives fully compliant with the standard SCRAPI programming interface for the RSVP protocol. We gave QTcl a high portability, in that it enables standard QoS negotiation to be performed in a seamless fashion on the most common operating systems.

SEKE '02, July 15-19, Ischia, Italy.

Keywords

Distributed Applications, Quality of Service, Programming Language.

1. INTRODUCTION

In the last few years, the availability of new communication technologies has brought to the development of a number of distributed multimedia applications. In particular, the availability of multicast support for IP applications has laid the grounds to the adoption of new paradigms for multimedia communication and Computer Supported Collaborative Work, within both local and wide area networks. However, this process has been highly chaotic, being focused more on satisfying specific needs than on designing open systems with high adaptiveness and interoperability features. A large number of applications, in particular multimedia applications, consist of a set of pre-existing components (building blocks) glued together by a common GUI. To develop applications of this kind, scripting languages have proved to be better suited than system programming languages [1].

Multimedia services impose a number of constraints on communication networks. Experiences over the Internet have shown the presence of a fundamental technical aspect: real-time applications do not work well across the network because of variable queuing delays and congestion losses. Before real-time applications can be broadly used, the Internet infrastructure must be modified in order to support more stringent QoS guarantees, mostly relating to the provision of some kind of control over end-to-end packet delays. Building global-scale distributed systems with predictable properties is one of the great challenges for computer systems engineering in the new century. Quality of Service requirements will be critical for distributed applications whose performance depends mainly on the characteristics of the communication service provided by the networking infrastructure [2].

Taking into account the current proposals stemming from the Internet research community, we envision a scenario where direct

interaction between applications and the underlying network infrastructure is definitely needed at least in the case of real-time communication. Hence, the need arises to define programming interfaces suitable for QoS-aware applications deployment in both the Integrated Services [3] and the hybrid (i.e. IntServ/Diffserv) IETF models. According to these models, applications can, with the help of an appropriate signaling protocol like RSVP (Resource reSerVation Protocol) [4], request communication services with per-flow bounds on communication throughput or end-to-end latency.

However, in spite of the impressive advance made in the design and implementation of performance guaranteed communication services, there has been lack of efforts in bringing such features at the application level in an organic approach. When developing distributed software with QoS awareness, programmers generally have to adopt ad-hoc solutions for specific operating systems and environments. We believe that this brings serious limitation to software reuse and portability and therefore that there is the need for a more flexible and open approach.

One possible alternative approach is to provide support for QoS into modern scripting languages, which are characterized by a broad adoption in the development of networked and distributed applications. In this paper we present QTcl, an extension of Cornell's Tcl-DP [11]. QTcl extends the Tcl-DP interpreter by providing a set of new commands, according to the standard SCRAPI application programming interface, defined by the IETF [5]. By exploiting the portability features of Tcl we wrote QTcl which represents the only available cross-platform SCRAPI implementation for the pursuit of quality of service on both Unix-based and Microsoft-based operating systems.

We feel that this effort for improving software development on top of advanced network architecture is only at the beginning and there is the need for the design of alternative and more open solutions. The goal of this paper is therefore to present our project rationale and the process that has brought to the development of QTcl.

The rest of the paper is organised as follows. In section 2 we briefly describe the QoS programming interface and in particular we focus the attention on a new programming interface made available for Microsoft Windows systems. In section 3 we present QTcl and the set of new commands that have been included in the language. In section 4 we illustrate how we have implemented QTcl, as an extension to Cornell's Tcl-DP. Section 5 presents a distributed VoD application where QTcl is used to protect data and control flows in a QoS-enabled internetwork. Finally, we discuss our conclusions in section 6.

The proceedings are the records of the conference. ACM hopes to give these conference by-products a single, high-quality appearance. To do this, we ask that authors follow some simple guidelines. In essence, we ask you to make your paper look exactly like this document. The easiest way to do this is simply to down-load a template from [2], and replace the content with your own material.

2. STANDARD QOS PROGRAMMING INTERFACES

IP has been playing for several years the most important role in global internetworking. Its connectionless nature has proved to be one of the keys of its success. Based on this assumption, the IETF Integrated Services working group has specified a control QoS framework in order to provide new applications with the appropriate support. Such a framework proposes an extension to the Internet architecture and protocols which aims at making broadly available integrated services across the Internet.

The key assumption on which the reference model for integrated services is built is that network resources (first of all its bandwidth) must be explicitly managed in order to meet application requirements. The overall goal in a real-time service, in fact, is that of satisfying a given set of application-specific requirements, and it seems clear that guarantees are hardly achieved without reservations. Thus, resource reservation and admission control will be playing an extremely important role in the global framework. The new element that arises in this context, with respect to the old (non-real-time) Internet model, is the need to maintain flow-specific state in the routers, which must now be capable to take an active part in the reservation process. RSVP protocol [4] is based on an exchange of messages according to the entity that supplies them (sender and receiver) or modifies them (intermediate network elements). The information supplied by each sender, and conveyed in the **PATH** messages, concerns the type of traffic that it is going to generate. Receivers send **RESV** messages which carry the service class to be used and the corresponding quality of service parameters.

In this framework, applications need to be modified in order to interact with QoS network. The IETF has defined RAPI [10], an application programming interface compliant with the RSVP Functional Specification. It is a user-level library written in C, which can be used by applications aimed at exploiting the QoS functionalities made available by a network reservation protocol like RSVP. RAPI calls let an application interact with a local RSVP daemon process, in order to establish a communication with QoS guarantees.

The RAPI interface is a first step towards the integration of communication services with QoS guarantees into applications; yet, its use is somewhat complex, since the application programmer must be aware of a number of parameters concerning the reservation. To cope with such problems, the IETF has proposed a simpler programming interface, layered on top of the RAPI and called SCRAPI [5]. SCRAPI provides three main functions:

Scrapi_sender, to be used by the sender of a data stream associated to an RSVP session,

Scrapi_receiver, to be used by the receiver, and

Scrapi_close, to close an RSVP session.

SCRAPI relieves users from taking care of all the details related to flow characterization, in terms of a formal token bucket description: the only required parameter is the average rate of the multimedia flow for which a reservation has to be enforced. Such a parameter might be retrieved in an automatic fashion in those cases where interaction with the service provider is made transparent to the lay-person . This can be achieved, for example, by exploiting a *metadata* repository associated to the multimedia content [13].

2.1 Interface W_SCRAPI

SCRAPI is only a programming interface to access the RSVP service, which must be implemented by a proper operating system module. In UNIX-like systems, this is usually a daemon process, which runs with root privileges in the end systems. For Microsoft Windows operating systems, the situation is a little bit more complex. QoS-functionalitiy and RSVP services are embedded in Win98. For WinNT, a distribution from Intel provides extensions to Winsock2 that are specific to RSVP protocol [14]. In Win2000, RSVP is seen as a particular service that might be launched. At the time of this writing, no implementation of the SCRAPI interface for these systems was available. To let developers be able to take advantage of QTcl in writing portable multimedia applications, we implemented a new interface for Microsoft Windows systems. In designing our interface, we followed the same philosophy that had led to the definition of the SCRAPI interface. Interface W_SCRAPI maps the SCRAPI calls onto the corresponding commands of QoS services on Windows based hosts. In other words, we ported the SCRAPI interface to Windows based systems.

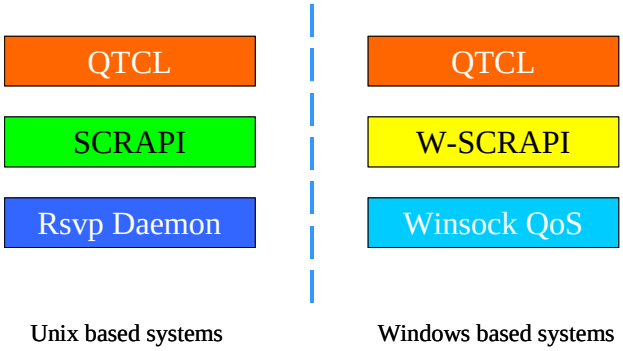

Fig.1 The SCRAPI interface in Windows based systems

W_SCRAPI is a library written in C, like the SCRAPI in UNIX based systems. It can be loaded as a dynamic link library when the Tcl shell is executed, and allows the execution of QTcl commands from Windows OS.

3. QTcl API

The SCRAPI programming interface has already been implemented as a C library, and used in modified Mbone tools [12]. However, the modifications were embedded in the application and thus multimedia application programmers are in charge to modify the original code in order to introduce QoS awareness. Our approach, instead, is related to the introduction of an intermediate layer responsible for the provision of QoS functionalities.

A support for QoS communication in Tcl scripting language was not available. Since we wanted to implement in Tcl a QoS-aware application for the distribution of multimedia documents, we have developed QTcl, an extension of the Tcl scripting language which implements the SCRAPI interface. QTcl provides the Tcl programmer with a set of new commands to create reservations in an RSVP-enabled network (which might be the access network in the hybrid scenario where IntServ and Diffserv infrastructres coexist). The new commands are shown below:

dp_scrapiSender dest_hostname
dest_port
source_hostname
source_port
bandwidth
protocol

dp_scrapiReceiver dest_hostname
dest_port
source_hostname
source_port
service
protocol

dp_scrapiStatus dest_hostname
dest_port
protocol

dp_scrapiClose dest_hostname
dest_port
source_hostname
source_port

Using these commands, it is possible to manage the whole process of reservation setup.

The bandwidth parameter must be expressed in Bytes/sec. The `service` parameter can be one of the following two values: `cl` indicating Controlled Load [7] or `gs` indicating Guaranteed Service [8]. Finally, the `protocol` parameter can be either `tcp` or `udp`.

`dp_scrapiSender` opens an RSVP session and starts **PATH** message transmission from source host to destination host. **PATH** messages are refreshed every 30 seconds.

`dp_scrapiReceiver` is invoked by a receiver in order to make a reservation request. The receiver specifies the desired QoS and class of service (Guaranteed Service or Controlled Load) according to the information contained into the **PATH** message. This request is forwarded to the sender across the network via a **RESV** message. After sending a **RESV**, the receiver waits for a confirmation of successful reservation from the sender for at most 10 seconds, as set by a specific timer; however, even in case of timer expiration the reservation process will go on.

`dp_scrapiStatus` allows to verify the current status of a session, according to a simplified error model available in the SCRAPI interface [5]

`dp_scrapiClose` is the function called to tear down an RSVP session, both in reception and in transmission

Figure 2 shows a simple application made of a sender process and a receiver process. The two processes should be executed on different hosts connected by an RSVP-enabled internetwork. The sender process invokes the `dp_scrapiSender` command, to start the transmission of **PATH** messages and then waits in a loop until the reservation is completed. The receiver process, instead, issues the `dp_scrapiReceiver` command to start the transmission of **RESV** messages and waits for the reservation to be completed. As soon as the reservation is setup, the sender starts transmitting UDP messages, 1480 bytes in length. The receiver, in turn, measures the time needed to receive a number N of such messages and estimates the received throughput. This simple

application can be tested in order to verify that the achieved throughput is independent of the network conditions, as long as the routers support QoS functionality.

Sender.tcl

```
# Sender
#!/home/qtcl/bin/tclsh8.0

package require dp

set sender [dp_connect udp -host 143.225.229.105\
   -port 3000 -myaddr localhost -myport 5000]

dp_scrapiSender 143.225.229.105 3000 \
   143.225.229.116 5000 100000 udp

while {$status != "green"} {
 after 1000
 set status [dp_scrapiStatus 143.225.229.105\
                              3000 udp]
}
puts $status

set pkt ""
for {set i 0} {$i < 1480} {incr i} {
 append pkt x
}

puts "Press ctrl-C to interrupt ...."

while {1} {
 set lun [dp_send $sender $pkt]
}

close $sender
```

Receiver.tcl

```
# Receiver
#!/home/qtcl/bin/tclsh8.0

proc bench { N } {
  global receiver
  set count 0
  while {$count < $N} {
    set rcv [dp_recv $receiver]
    incr count [string length $rcv]
  }
}

package require dp

set receiver [dp_connect udp -myport 3000]
fconfigure $receiver -blocking 1

dp_scrapiReceiver 143.225.229.105 3000 \
              143.225.229.116 5000 gs udp

while {$status != "green"} {
 after 1000
 set status [dp_scrapiStatus 143.225.229.105\
                              3000 udp]
}
puts $status

set N  10485760
set T  [lindex [time { bench $N }] 0]
set BW [format "%2.3f" [expr $N*8.0/$T]]

puts "Elapsed time: $T microseconds"
puts "Estimated bandwidth: $BW Megabit/sec"

close $receiver
```

Fig. 2 An example of QTcl commands use

4. QTcl implementation

QTcl has been conceived as a tool for supporting the development of distributed applications with simple QoS requirements. In order to minimize the development effort, we tried to exploit some useful features that were already available in the Tcl-DP extension, developed at Cornell University [11]. In particular, we found the `dp_RPC` mechanism particularly suitable to support the receiver-initiated reservation mechanism of RSVP. Hence, QTcl has been developed starting from the original Tcl-DP source distribution. We then extended the Tcl interpreter by creating a set of C functions that implement the SCRAPI primitives.

We briefly describe the steps we did to add the new command `dp_scrapi_Sender` to Tcl-dp. First of all, the presence of this new command has to be defined. That was done by modifying these two files.

In file dpInit.c, we defined the dp_scrapiSender command and the related procedure:

```
static DpCmd commands[] = {
…
{"dp_scrapiSender",
      dp_createSender},
…
}
```

In file dpInt.h, we wrote the prototype dp_createSender:

```
EXTERN int dp_createSender
_ANSI_ARGS_((ClientData
clientData,Tcl_Interp *interp, int argc,
char **argv));
```

Finally, we implemented new procedure in a new file (that we called scrapi_tcl.c)

```
/* Prototype of new Tcl commands */

int dp_scrapiSender (Tcl_Interp
*interp);
…
/* Procedure defined in this module */

int dp_createSender(ClientData
clientData,
                    Tcl_Interp
*interp, int argc, char **argv);
…
/* dp_createSender implementation */
int dp_createSender(ClientData
clientData,
                    Tcl_Interp
*interp, int argc, char **argv)
{
…
}
```

Last step has been the upgrade of Makefile.in in order to take into account the new file scrapi_tcl.c:

```
….
OBJS = \
       $(OBJ_DIR)/dpChan.o \
       $(OBJ_DIR)/dpCmds.o \
          $(OBJ_DIR)/scrapi_tcl.o \
       $(OBJ_DIR)/dpInit.o \
…
```

In the next section we will describe some trials we carried out using an heterogeneous scenario.

5. A QOS-AWARE DISTRIBUTED MULTIMEDIA APPLICATION BASED ON QTCL

To show how effective QTcl can be in distributed software development and the easiness of RSVP bandwidth management in a real application, we used the extended scripting language to add the ability of making network resource reservations to DiVA, a distributed multimedia application for cooperative video distribution over the Internet, developed by our research group. DiVA is capable of playing and controlling remote audio/video documents over a community of users in a synchronized way in streaming mode. We gave DiVA both the capability to dynamically adapt to current network conditions and to actively interact with QoS-capable network architectures (via standard APIs) in order to guarantee the users a specified level of service.

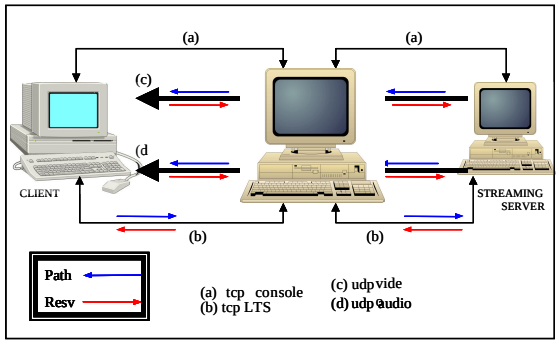

Fig. 3 Data streams generated by the DiVA application and associated to RSVP sessions.

DiVA is a complex multimedia application, since it involves the transmission of multiple data flows conveying the content, control commands and synchronization information, according to a layered software architecture. Figure 3 shows the relevant data streams produced by the DiVA application between a streaming server host and a client host. In particular, the UDP audio and video streams are transmitted downstream from the server on the right to the client on the left, while two TCP bi-directional streams are used to exchange control (console) and synchronization (LTS – *Logical Time System*) information. [15]

Our approach is to exploit the capability of QTcl as a scripting language to have all the issues related to resource reservation for the different media solved according to a broker-based approach, in which all the interactions between the application and the network are performed by a new, independent software module added to the application client interface. According to this approach, it was required no intervention in the software modules involving both data and control communication. When the application client is launched, the new module for resource reservation takes care of interacting with the network infrastructure: because RSVP protocol is receiver initiated, the reservations are made by the clients on the basis of the sender specifications. Hence, server applications are in charge of starting the communications by sending the PATH messages with traffic envelope information. Our purpose was to let this procedure become automated and seamless to the users. To do that, we fruitfully used the RPC call available from Tcl-DP: in the following lines, we report some Tcl code from the DiVA client application that shows how reservations are done automatically on the basis of default values.

```
# RPC to request PATH messages from the server
 # Wait for .1 sec
 after 100
 if [catch {dp_RPC $sockV -timeout 60000 ScrapiSndPath
$C_UdpSrcV $C_UdpDestV $bwVideo $service} error] {
     # server error request
     catch {diva_CloseRPC $sockV}
     error "Unable to have full server connection. \
     \nconnection reports: $error"
 }
……..
 # Send reservation request to the server
 after 100
 dp_scrapiResv [lindex $C_UdpDestV 0] [lindex $C_UdpDestV
1] [lindex $C_UdpSrcV 0] [lindex $C_UdpSrcV 1] $service udp
```

In these code lines, the ScrapiSndPath call is a RPC to a server side procedure that causes the forwarding of PATH messages with video traffic specifications. Then, by using the dp_scrapiResv procedure the client application can eventually complete the reservation with correct parameters.

We tested the application in a testbed formed by two different Local Area Networks, connected by means of a WFQ router implemented in FreeBSD [6]. The router was connected to the first LAN through a 100 Mb/s Fast Ethernet card and to the second LAN through a 10 Mb/s Ethernet card. A host in the 10 Mb/s LAN acted as a client, while another host in the 100Mb/s LAN ran the DiVA video server. Hence, multimedia traffic flowed through the WFQ router.

As we already mentioned, in our prototype, the bandwidth values used to setup reservations for the video and audio streams were determined empirically for each archived document, by observing the traffic produced by the application while streaming it- These values were provided by the final users to the DiVA client software. In a real-world application, however, it might be logical to expect that users are not aware of the QoS requirements of multimedia documents. We expect therefore that these values might be retrieved automatically by the client application in the form of *metadata* associated to the document, as in the GESTALT architectural model [9].

6. DISCUSSION AND CONCLUSIONS

An increasing number of distributed applications can benefit from the availability of improved communication services in RSVP-enabled IP internetworks, by acting in a proactive way, instead of passively adapting to the available QoS offered by current best-effort services. We believe that this support is helpful for a wide range of modern distributed applications. In this paper we have presented QTcl, a QoS control API which is compliant with the IETF SCRAPI interface, and has been designed as an extension of the Tcl scripting language.

QTcl appears to be a good tool for the development of novel, QoS aware applications: in fact, extending a well established language appears to be a good initial solution for the problem of developing distributed software suited for the exploitation of new network infrastructures. The proposed extensions try to hide as much as possible to the programmer the need for a detailed knowledge of the technicalities associated to the reservation of network resources according to the resource control model available in the communication architecture. We have shown that it is possible to extend an existing distributed multimedia application by adopting a broker-based approach, where the inclusion of a new module enables the application both to allow to interact with QoS aware networks and to avoid extensive intervention on the existing code.

It is clear, however, that with the large deployment of these infrastructures, it will be necessary to follow a different approach, consisting in the semantic extension of new languages towards the issues related to network programming with guaranteed communication performances.

We believe that QTcl represents an initial, concrete answer to the request of programmers of real-time and multimedia distributed applications, as it has been shown by presenting a video distribution application developed with our scripting language.

7. REFERENCES

[1] J.K. Ousterhout. "Scripting: Higher-Level Programming for the 21st Century". Computer, March 1998, pp.23-30.

[2] K. Kavi, J.C. Browne, and A. Tripathi. "Computer Systems Research: The Pressure Is On". Computer , Jan. 1999, pp. 30-39.

[3] R. Braden, D. Clark, and S. Shenker. "Integrated Services in the Internet Architecture: an Overview". IETF RFC 1633, July 1994.

[4] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. "Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification". IETF RFC 2205, September 1997.

[5] B. Lindell. "SCRAPI - A Simple 'Bare Bones' API for RSVP". IETF Internet Draft draft-lindell-rsvp-scrapi-02.txt, Feb. 1999.

[6] R. D'Albenzio, S. P. Romano and G. Ventre. "An Engineering Approach to QoS Provisioning over the Internet". Lecture Notes in Computer Science no. 1629, Springer, May 1999, pp. 229-245.

[7] J. Wroklawsky. "Specification of the Controlled-Load Network Element Service". IETF RFC 2211, Sep. 1997.

[8] S. Shenker, C. Partridge, and R. Guérin. "Specification of Guaranteed Quality of Service". IETF RFC2212, September 1997.

[9] The GESTALT Project, http://www.fdgroup.co.uk/gestalt/

[10] R. Braden and D. Hoffman. "RAPI -- An RSVP Application Programming Interface - Version 5". IETF Internet Draft draft-ietf-rsvp-rapi-01.txt, Aug. 1998.

[11] M. Perham, B. C. Smith, T. Jánosi, and I. K. Lam. "Redesigning Tcl-DP". Procs. of the Fifth Annual Tcl/Tk Workshop, Boston, 1997.

[12] USC Information Sciences Institute (ISI), http://www.isi.edu/rsvp/release.html

[13] S. P. Romano, S. Russo, G. Ventre and P. W. Foster, “An architecture for wiring QoS requirements into multimedia data”**,** submitted to the *World Wide Web journal.*

[14] http://developer.intel.com/ial/rsvp/

[15] B. C. Smith, L. A. Rowe, J. A., Konstan, and K. D. Patel, “The Berkeley continuous media toolkit”, In Proc. of the 4th ACM International Multimedia Conference, Boston, November 1996.