Bringing Service Differentiation to the End System

Domenico Cotroneo, Massimo Ficco, Simon Pietro Romano and Giorgio Ventre

Via Claudio, 21
Dipartimento di Informatica e Sistemistica
Università degli Studi di Napoli "Federico II"
80125 Napoli, Italy
{cotroneo,spromano,ventre}@unina.it ficcomax@grid.unina.it

Abstract.

A number of distributed applications require communication services with Quality of Service (QoS) guarantees. To the purpose, resource management is required both in the end-system and inside the network.

To date several approaches have been proposed, dealing either with end-system issues or with those more strictly related to the network, but no unified view exists. Furthermore, while network QoS provisioning has come to a good level of standardization, no standard proposals exist for the end-systems. We argue that a single architectural model is needed, taking into account a more exhaustive concept of "resource management", where quality of service is defined at the level of user perception.

In this paper we propose an architectural model that enhances the IETF Diffserv framework in order to provide service differentiation even inside network end-points.

We also present a prototype implementation of such architecture along with experimental results validating the proposed solution.

Index terms — **Quality of Service, Operating Systems, Differentiated Services**

A. INTRODUCTION

To provide applications with end-to-end guarantees, network resource management alone is not sufficient. This is particularly true when end-points become more sophisticated, e.g., workstations with rich device support, multiprocessing and multiple users. Distributed applications usually require a diversity of resources and network bandwidth is just part of them. Other features, such as processing capacity, have to be managed in concert with the network to provide applications with a guaranteed behavior [1].

With respect to the network, experiences over the Internet have showed the lack of a fundamental technical element: real-time applications do not work well across the network because of variable queuing delays and congestion losses. The Internet, as originally conceived, offers only a very simple quality of service: point-to-point best-effort data delivery. Thus, before real-time applications can be broadly used, the Internet infrastructure must be modified in order to support more stringent QoS guarantees.

Based on these assumptions, the *Internet Engineering Task Force (IETF)* has defined two different models for network QoS provisioning: *Integrated Services* [2] and *Differentiated Services* [3]. These frameworks provide different, yet complementary, solutions to the issue of network support for quality of service.

Although network designers stress the point that the network is more than its applications, we argue that the applications themselves drive its evolution and deployment. A single architectural model is thus needed, taking into account a more exhaustive concept of "resource management", where the quality of the service is defined at the level of user perception.

We propose a framework that extends the concept of service differentiation, introducing a schema for priority-based communication mechanisms inside the end point operating system. We also present a testbed implementation of this framework running under Solaris 2.6 and provide a critical discussion of the main experimental results.

The rest of the paper is organised as follows. In section B we briefly introduce the QoS provisioning issue in the Internet, taking a look at the Differentiated Services model proposed by the IETF. In section C we present an overview of some interesting proposals for end-to-end QoS support inside the end system.

An innovative proposal for end-system resource management is depicted in section D, where we introduce a new component called the *Priority Broker* (PB). We describe the overall structure of our enhanced differentiated services end-to-end architecture in section E. In section F we illustrate implementation issues related to the Priority Broker, while presenting experimental results in section G.

Conclusions and directions for future work are provided in section H.

B. NETWORK QOS

In the past several years work on scalable QoS-enabled networks has led to the development of the Differentiated Services *(Diffserv)* Architecture. A Diffserv network achieves scalability by means of aggregation of traffic classification state conveyed in an appropriate DS field set by IP-layer packet marking [4].

In the Diffserv framework traffic entering the network is subjected to a conditioning process at the network edges and then assigned to one of different behavior aggregates, identified by means of a single DS codepoint. Furthermore, each DS codepoint is assigned a different per hop behavior, which determines the way packets are treated in the core of the network. Thus, differentiated services are achieved through the

0-7695-0777-8/00 $10.00 © 2000 IEEE

appropriate combination of traffic conditioning and per-hop behavior forwarding. To date, the most interesting proposals are an Expedited Forwarding (EF) [11] per hop behavior and an Assured Forwarding (AF) [10] per hop behavior.

C. END SYSTEM QOS

A number of researchers dealt with the issue of providing Quality of Service guarantees inside network end-points.

David K.Y. Yau and Simon S. proposed an end system architecture designed to support network QoS [5]. They implemented a framework, called Migrating Sockets, providing performance-critical protocol services, such as send and receive, via a user level library linked with applications.

An architectural model providing Quality of Service guarantees to delay sensitive networked multimedia applications, including end systems, has been proposed by K. Nahrstedt and J. Smith [6], who also designed a new model for resource management at the end-points, called the QoS Broker [7]. The broker orchestrates resources at the end-points, coordinating resource management across layer boundaries. As an intermediary, it hides implementation details from applications and per-layer resource managers.

While we recognize that the cited works represent fundamental milestones for the pursuit of quality of service mechanisms inside the end-systems, we claim that the time is ripe for taking an integrated view on QoS-enabled communication.

A single architectural model is needed, integrating network centric and end-system centric approaches and taking into account the most relevant standards proposed by the research community.

D. A NEW PROPOSAL: THE PRIORITY BROKER

A number of research studies [1] [5] show that the operating system has a substantial influence on communication delays in distributed environments. In particular, packet scheduling policies inside the operating system turn out to be a crucial issue for the provisioning of some guarantees which are of paramount importance for end-to-end QoS enforcement.

To the purpose, we developed a schema for priority-based communication mechanisms inside the operating system, which we called the *Priority Broker (PB).*

The PB is an architectural model conceived to support communication flows with a guaranteed priority level. It is implemented as a new software module residing between applications and the kernel and providing an additional level of abstraction.

Figure 1 shows a conceptual view of the Priority Broker.

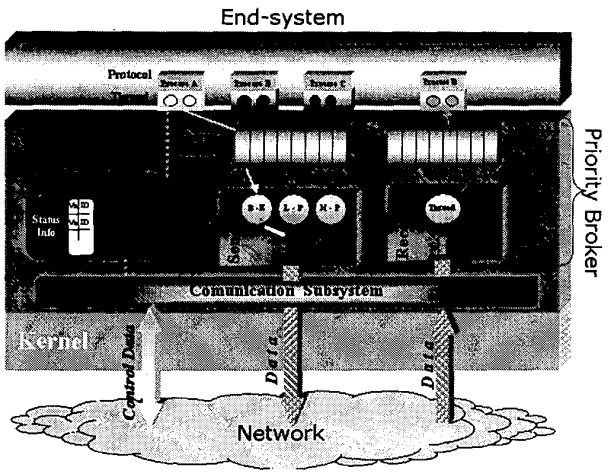

Figure 1. Simplified model for the Priority Broker

The PB offers a number of communication services, belonging to the following three classes:

- **best-effort:** no QoS guarantees;
- **low priority:** for adaptive applications;
- **high priority:** for applications that are most demanding in terms of QoS guarantees.

The new module is part of a software architecture capable of interacting with the operating system in order to provide the applications with a well-defined set of differentiated services, while monitoring the quality of the communication by means of flow control techniques.

Four are the major components that build the broker (see Figure 1): the *sender-broker,* the *receiver-broker,* the *communication sub-system,* and the *connection manager.*

The sender-broker is responsible for the transmission of packets with the priority required by end-users, while the receiver-broker manages incoming packets. The communication subsystem provides the I/O primitives that allow discriminating outgoing packets in the kernel I/O subsystem. Finally, the connection manager is in charge of handling all service requests on behalf of the local and remote processes.

To provide QoS guarantees, the Priority Broker exploits the following mechanisms:

- *I/O primitives* made available by the Operating System to assure flow control on the sender side when accessing the network;
- *Shared memory* to avoid multiple copies between applications and kernel to send and receive messages;
- *Synchronization mechanisms* to implement the communication protocol between user applications and PB;
- *Demultiplexing capabilities* on the receiving side.

Our Priority Broker has been conceived as a daemon process running on the end-systems. Communication between users' applications and the broker is made possible by an ad hoc defined library, which we called the *Broker Library Interface (BLI).*

The PB, in turn, relies on the services made available by both the *transport provider* and the I/O mechanisms of the operating system upon which it resides.

E. END TO END QOS PROVISIONING

We propose here an architecture that enhances Diffserv communication paradigm in order to bring the concept of differentiation even to the end-system, putting aside the simplistic view that looks at it as an atomic point with no specific requirements in terms of resource allocation and control.

Thus, we envision a distributed environment where resources are explicitly required by the applications and appropriately managed by the local OS, which is also in charge of mapping local requests onto the set of services made available by the network.

More precisely, we enforce a correspondence between end-system priorities and network classes of service, thus obtaining an actual end-to-end Per Hop Behavior, where the first and last hops take into account transmission between end hosts and network routers.

Communication in this scenario takes place with the following steps:

1. A sender application chooses its desired level of service among the set of available QoS classes and it asks the local connection manager to appropriately setup the required resources.
2. The connection manager tests for local admission control and, in case of success, sets the particular DS codepoint corresponding to the required class of service. To cope with this issue, an appropriate mapping must be defined between local classes of service and network QoS levels. The solution we propose is shown in the following table:

Local priority	Network service
High	EF
Low	AF
Best Effort (no priority)	Default

F. PB IMPLEMENTATION ISSUES

The prototype Priority Broker we implemented is a software module placed between application and kernel space. It provides a set of communication services, by means of a library (BLI, *Broker Library Interface*), allowing applications to request a communication channel with a fixed priority .

Next sub-sections describe in detail the PB components mentioned in section D, and how they interact in an Inter-PB communication scenario.

We implemented the PB on Sun Solaris 2.6 operating system.

1st. Connection Manager

The connection manager is the PB component in charge of handling all service requests on behalf of local and remote processes. It provides the following services:

- control and data sockets management;
- service requests scheduling on behalf of remote processes;
- control information handling;
- I/O buffers handling;
- local database management.

This component also manages a local database containing information needed for communication between local and remote processes.

At startup a local process requests a communication service, with a fixed priority, to the local PB, specifying the remote host it wishes to communicate with. A control socket with the remote PB is created and a request message is sent to the remote PB connection manager.

Once the request message has been accepted, the remote PB connection manager stores a new record into its local database and sends back an acknowledgement message to the source PB, indicating the Virtual Name of the remote process.

Upon reception of this acknowledgement message, the source PB creates a data socket and appropriately configures its Diffserv marker so to map the local priority level onto the right level of service adopted over the Diffserv communication network.

2nd. Sender subsystem

The Sender subsystem provides the following services:

- *Inter-PB data unit creation.* An Inter-PB data unit is a unit of information that travels from the sender PB to the receiver PB. This data unit is created by adding a header to the process message. Such header contains two fields: the destination process ID, and the message dimension.
- *Inter-PB data unit transmission with the requested priority.* The PB sends data units using the communication channel made available by the Connection Manager.

It is structured as a multithreaded process that manages a pool of LWP threads [8]. Each priority level, requested form local processes, is associated to one thread. Each thread is in charge of transmitting data units with the assigned priority, relying on the set of communication services defined in the BLI (Broker Library Interface) library.

Data exchanging between transport and application layers is obtained via a shared memory, thus avoiding unnecessary copies. Each process connecting to the PB is assigned a portion of such memory area, where all data coming from the Receiver Broker are placed.

In order to achieve synchronization between user processes and the Sender, we adopted a "1 producer n consumers" paradigm, which we implemented by means of mutex[9].

3rd. Receiver subsystem

The Receiver Broker accomplishes the following tasks:

- *Message reconstruction.* It extracts sender messages from the packets received from the data socket (which contain an additional header)
- *Message demultiplexing.* It delivers received messages to the appropriate processes.

The Receiver broker is implemented as a unique LWP thread, taking into account all of the defined priority levels.

4th. Communication subsystem

As a general remark, it is obvious that the features of the communication subsystem are heavily dependent on the particular operating system adopted. For our implementation we chose to exploit a general purpose OS.

Our PB runs under Solaris 2.6, which provides a powerful I/O mechanism, the STREAMS framework [9]. This mechanism consists of a set of system calls, kernel resources, and kernel routines. The fundamental STREAMS unit is the Stream. A Stream is a full-duplex bi-directional data-transfer path between a process in user space and a STREAMS driver in kernel space. It is composed of a Stream head, zero or more modules, and a driver. Data on a Stream are passed in the form of messages. Each Stream's module has its own pair of queues, one for reading and one for writing. Queues are the basic elements by which the Stream head, modules, and drivers are connected. When called by the scheduler, a module's service procedure processes queued messages in a first-in, first-out (FIFO) manner, according to priorities associated with them. Thus, STREAMS uses message queuing priority. All messages have an associated priority field: default messages have a priority of zero, while priority messages have a priority band greater than zero. Non-priority, ordinary messages are placed at the end of the queue following all other messages that can be waiting. Priority band messages are placed below all messages that have a priority greater than or equal to their own, but above any with a smaller priority[9].

To discriminate among three different classes of service, we found, by trial and error, that the best choice was to use a priority of 0 for Best Effort, 1 for Low Priority and 255 for High Priority.

5th. BLI Library

The *Broker Library Interface (BLI)* acts as an interface between applications and the Priority Broker. It makes available to the end-user a set of primitives that provide the applications with the necessary support for differentiated communication services. Two are the major entities involved in the communication: the *broker endpoint* and the *broker provider.* The former represents either local or remote process, whereas the latter is the set of routines implementing the PB upon the end-system.

Using BLI functions, an application may be granted access to the services provided by the Priority Broker.

Figure 2 illustrates a functional schema describing the client-server communication protocol implemented via the BLI.

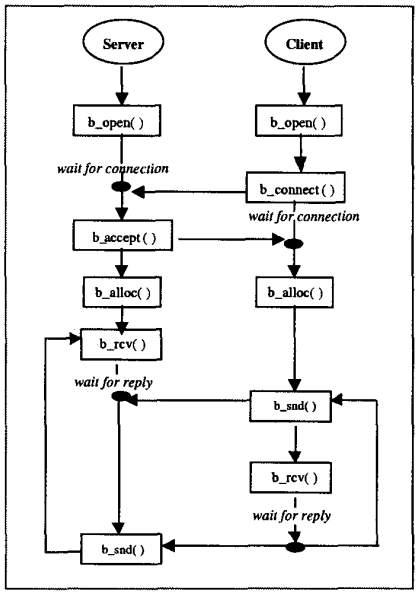

Figure 2. Client-server communication via the BLI

As depicted in the figure, the first step required is the connection with the local PB (b_open() call both on client and on sender side). Afterwards, a client invokes b_connect() to ask a server application for a specific service. After accepting such request (b_accept() on the server side), the next step is the invocation of b_alloc(), which results in memory allocation for both reading and writing buffers.

Finally, data transmission and reception is achieved via b_snd() and b_rcv() invocations.

G. EXPERIMENTAL RESULTS

In this section we present some of the experiments we made in order to validate the behavior of our Priority Broker and we provide a critical analysis of the most significant results we obtained.

Test 1 – Service guarantees provided by the PB

The first experiment was run to investigate the kind of guarantees the Broker is able to provide.

A server application was run on one workstation, whereas two client applications were run concurrently on another workstation. Both clients were connected to the server through PB communication services. The former (A) asked for a Best-Effort treatment, while the latter (B) required a High Priority service. Both clients were sending 20000 messages, each 1200

bytes long. Communication was activated first by client A; client B came into play some seconds afterwards.

Figure 3 shows the results obtained with this experiment.

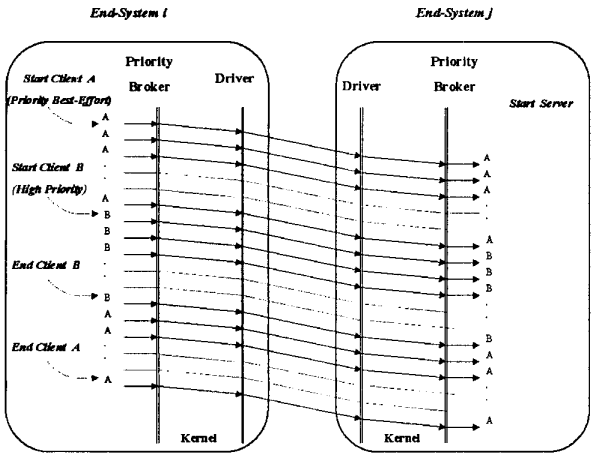

Figure 3. Results of the first experiment

It is worth observing that client A transmission is preempted as soon as client B starts sending packets. Transmission from A is resumed only when B stops.

Test 2 – Broker's degree of intrusion on standard communication

A testbed was set up to evaluate the impact of the Broker module on applications using standard communication libraries, while running concurrently.

Bandwidth measures show that the increase in the number of clients exploiting PB services (varying-blue columns) has only a small effect on the resources assigned to the client that uses standard sockets (red column).

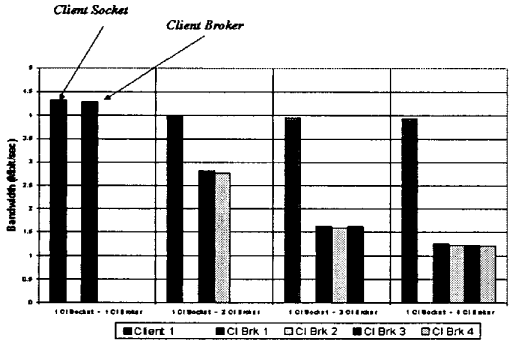

Figure 4. Results of the third experiment

This is due to the fact that the processes represented by blue columns in the figure belong to the same stream, that has been created by the broker; thus, they do not affect that much the standard sockets client.

H. CONCLUSIONS

In this paper we discussed some of the issues involved in the design of a framework that extends the concept of service differentiation. We proposed an architecture that enhances Diffserv communication paradigm in order to bring the concept of differentiation even to the end-system, putting aside the simplistic view that looks at it as an atomic point with no specific requirements in terms of resource allocation and control.

Thus, we introduced a schema for priority-based communication mechanisms inside the end point operating system. We also presented a testbed implementation of this framework and provided a critical discussion of the main experimental results.

Work is in full swing to integrate our Priority Broker component in the framework of a Diffserv network testbed we set up at our Laboratory. Ongoing experiments are dealing with the refinement of the interaction between the PB and the edge routers of the Diffserv network and with the validation of the mapping process we implemented..

Our aim is to design a distributed environment where resources are explicitly required by the applications and appropriately managed by the local OS, which is also in charge of mapping local requests onto the set of services made available by the network.

I. REFERENCES

[1] Henning Schulzrinne, "Operating System Issues for Continuous Media," Multimedia Systems, vol. 4, no. 5, pp. 269--280, Oct. 1996.

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

[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. "An Architecture for Differentiated Services". RFC 2475, Dec. 1998.

[4] K. Nichols, V. Jacobson, and L. Zhang. "A Two-bit Differentiated Services Architecture for the Internet". Internet draft, draft-nichols-diff-svc-arch-00.txt, Nov. 1997.

[5] David K.Y.Yau and Simon S. Lam "Migrating Sockets-End System Support for Networking with Quality of Service Guarantees", IEEE/ACM Trans. On Networking. VOL. 6, December 1998.

[6] K. Nahrstedt and J. Smith "*A Service Kernel for Multimedia Endstations*", Technical Report, MS-CIS, University of Pennsylvania.

[7] K. Nahrstedt and J. Smith, "The QoS Broker", IEEE Multimedia Spring 1995,Vol.2, No.1, pp. 53-67.

[8] John R. Graham, *Solaris 2.x, internal and architecture*, Mc. Graw Hill, 1995.

[9] Uresh Vahalia, *UNIX Internals, The new frontiers*, Prentice Hall International, 1996.

[10] J. Heinanen, F. Baker, W. Weiss and J. Wroclawski. "Assured Forwarding PHB Group". Internet draft, draft-ietf-diffserv-af-04.txt. January 1999

[11] V. Jacobson, K. Nichols, K. Poduri. "An Expedited Forwarding PHB". Internet draft, draft-ietf-diffserv-phb-ef-01.txt. November 1998