An Inter-application and Inter-client Priority-based QoS Proxy Architecture for Heterogeneous Networks

Qiang Wang, Qing Ye, Liang Cheng
Laboratory Of Networking Group (LONGLAB, http://long.cse.lehigh.edu)
Department of Computer Science and Engineering, Lehigh University, Bethlehem, PA, USA
qiw3@lehigh.edu, qiy3@lehigh.edu, cheng@cse.lehigh.edu

Abstract

Most QoS (Quality of Service) adaptation systems in a heterogeneous network environment require source-code modifications in order to implement adaptation mechanisms, and network QoS parameters (e.g. bandwidth, latency, and jitter) are major sources for adaptation. This paper proposes a three-tier QoS proxy architecture where legacy applications can be transparently integrated without source-code modifications. The other salient feature of the architecture is its comprehensive coverage of QoS parameters for adaptation, including network QoS parameters such as bandwidth, latency and jitter, system QoS parameters such as CPU load and battery power status, and user profiles and QoS preferences. An end-user may assign different priorities to different applications through graphical user interfaces to best fit their QoS requirements. Packets from a client application are differentiated at the proxy, which integrates the inter-application and inter-client priority policies using priority fair queues.

1. Introduction

Heterogeneity is an important characteristic of computing environments nowadays. A heterogeneous computing environment consists of a number of dissimilar hardware and software elements, e.g. different network connections, devices, and services. For example, in a university campus, the network connections may comprise of wired and wireless ones, which may include: slow dial-ups from student dorms and high-speed links from computer labs. The physical computing devices may encompass a mix of personal computers, handhelds, smart appliances, and even tiny sensors; and applications having different quality of service (QoS) requirements may include bandwidth-demanding P2P file sharing, computation-intensive scientific simulation, real-time multimedia streaming, etc. To provide ideal service quality for all users and applications in such an environment, heterogeneity needs to be comprehensively addressed and QoS infrastructure be carefully designed. How to address the QoS issues in the context of the heterogeneous environments becomes a challenge.

Several approaches have been researched and implemented to meet this challenge, including various designs of system architectures, service protocols, and control algorithms. One approach is resource reservation. In a reservation-based system (e.g. an ATM network), the system will dedicate necessary resources to an application to meet the QoS requirement. However, a reservation-based system needs network infrastructure supports. Adaptation approach, in contrast to the reservation approach, generally occurs at the application level and offers more controllable application-specific adaptation choices. An adaptive application does not require a tight integration or modification of the best-effort service provided by the traditional networks and operating systems' protocol stacks. However, adding either adaptation or resource-reservation functionality to an application generally needs source code modification, which is not accessible for an enormous number of legacy commercial programs in usage. In our research, we propose a proxy-based architecture to provide the QoS support, which is transparent to existing applications.

Proxies have become more and more prevalent in recent years in corporate and academic networks. Proxies are used to process data flowing between two end hosts as an intermediary. For clients, a proxy acts as their servers; while for servers, a proxy is a special client, which intermediates data communication for clients. Proxies are generally used to provide efficient use of network resources, reduced cost, and increased

Proceedings of the 10th IEEE Symposium on Computers and Communications (ISCC 2005)
1530-1346/05 $20.00 © 2005 **IEEE**

security. There are research efforts on QoS proxy to support multimedia services [13] and communications across wireless links [14]. However, legacy applications need source-code modifications to be compatible to the QoS architecture proposed in [13] considering its unified component-based programming environment. Only wireless link bandwidth and user profile and feedback are considered by the QoS management architecture proposed in [14]. In this research, we extend the environment-awareness capability of the QoS proxy to user/application QoS parameters, system QoS parameters, and network QoS parameters. Moreover, the QoS-support architecture proposed here helps applications be transparently deployed in a heterogeneous network environment without the need of source-code modifications and be adaptive to the possible variations of networks, systems, and end users' preferences. Next section discusses related work and Section 3 presents our QoS proxy architecture. Section 4 concludes this paper.

2. Related Work

Bolliger and Gross [4] have proposed the concept of network-aware as the ability of applications to adjust their resource demands in response to the network performance variations. Network characteristics such as bandwidth, latency, and jitter are QoS parameters for the adaptation. Application-aware concept is defined as a collaborative partnership between the operating systems and applications to offer a general and effective approach to network information access. The concept of context-aware in [5] is similar to that of environment-aware proposed in [6]. Both are confined to the awareness of changes between end hosts and network environments.

There exist research of QoS-aware architecture in a middleware design of multimedia network applications [7][9][13]. In this paper, the concepts mentioned above are extended to QoS-aware, which is defined as the system's ability to deal with and adapt to the changes of the user-defined QoS profiles, end-host resource availability, and communication network characteristics. The ultimate goal of QoS-aware architecture is to achieve the best possible service quality specified by users and the most efficient resource utilization constrained by system features.

Table 1 compares several QoS adaptation architectures that exemplify existing work. Our research is unique in two aspects. One is that our QA-Proxy (QoS Adaptation Proxy) based architecture enables a transparent integration with legacy applications without the requirement of source code modification. The other is its comprehensive coverage of QoS parameters for adaptation. Existing QoS architectures focus on part(s) of network QoS parameters such as bandwidth, latency and jitter, system QoS parameters such as CPU load and battery power status, and/or user profiles. We consider the user's QoS parameters by letting users select the priority of a program that suits their QoS needs. The network dynamics and devices characteristics are considered in our architecture as well.

Table 1. Network QoS adaptation architectures

System	Adapt-ations at	Sources for adaptations (QoS data)	Code change?	Control algorithm for adaptation
Odyssey [10]	Client side	Bandwidth, latency, battery power, CPU utility, memory size	Yes	Each type of QoS data has a specific warden that is in charge of retrieving data fidelity in response to the variation of system resources
Chariot [4]	Server side	Bandwidth	Yes	Forming a close-loop control based on the difference between time needed to deliver a response with actual time left to deliver it
ACAN [5]	Server and client side	Bandwidth	Yes	Using mobile agents to detect and transmit network information and perform adaptation
Network Weather Service [11]	Server side	Bandwidth, CPU usage, latency, memory size	Yes	Detecting the current resource usage of the system and forecasting the future dynamic variation to help clients choose an appropriate server
Agilos [8]	Middle-ware located at server and client sides	Bandwidth	Yes	Using resource observer and adaptor to form a close-loop control. An adaptor controls applications to choose an appropriate form of components to adapt to the environment.
Our QA-Proxy System	Proxy based	Network, system, and user preference awareness	No	Calculating the packet priority based on the application priority assigned by users and the client's priority assigned based on the client's profile.

3. QoS-aware Architecture

The usage scenarios of our architecture are within networks where the client/proxy/server access model is prevalent. For instance, shown in Fig. 1, each client's traffic goes through a QA-Proxy to access the Internet; the traffic flows from the Internet are dominated by HTTP, multimedia, and FTP data, etc; and the traffic inside the intranet is generally for accessing of file repositories, web servers, and database servers. We construct a three-tier QoS architecture where the QoS-Adaptation Proxy (QA-Proxy) implements adaptation mechanisms, which should otherwise be done by client-side applications. In other word, the QA-Proxy takes the responsibility of the adaptation and the client applications remain the same without source code modifications. Clients send data to QA-Proxy, which will forward the requests to destination servers based on QoS mechanisms explained in later subsections. Server responses are cached first in the proxy and then be sent back to an originating client by the proxy, also with QoS mechanisms performed during the process.

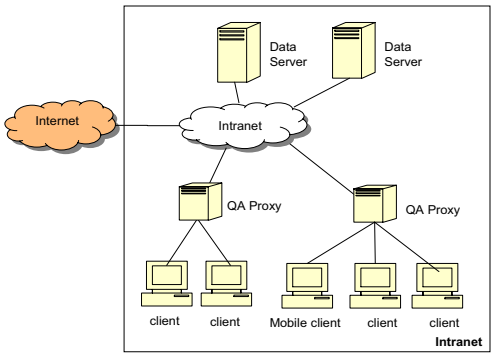

Figure 1. An example of a usage scenario of the QoS Adaptation Proxy (QA-Proxy) architecture

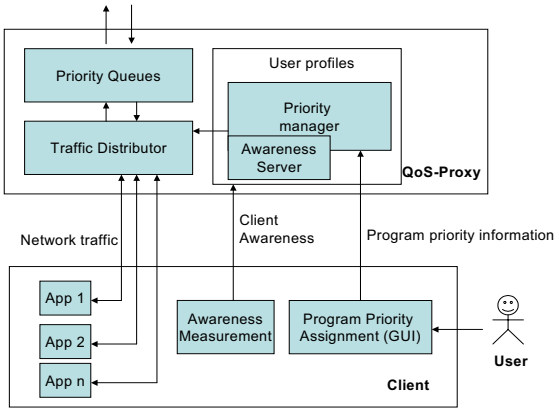

Figure 2. Major components of the QoS architecture

Fig. 2 illustrates individual components in the QoS architecture. At each client, there are two major components: Awareness Measurement Module and Program Priority Assignment Interface. Both of them are used to acquire the QoS data of the client: awareness measurement tools use active probing approach to collect network and device information, e.g. available bandwidth to the proxy and battery consumption at wireless devices. The Program Priority Assignment Interface is a graphic user interface, through which a user is able to specify the relative priority of specific programs. The following subsections will describe each component in details.

3.1. Network and System QoS Specification

QoS specification at end hosts is to retrieve end-system QoS data and user-preference QoS data. By collaborating with the operation system (OS), awareness measurement tools collect end-system QoS data (e.g., CPU load and memory usage) by invoking system APIs and integrating with existing tools. For example, existing tools like vmstat and uptime [11] probe real-time workload of CPU and available memory under a Linux OS.

Getting accurate network condition information is essential to the network adaptation in general and to our QoS architecture in specific. Our network measurement tools have been reported in [1] [2] and [12] and detailed explanations of the theoretical and practical issues about these tools are beyond the scope of this paper. In short, [2] provides an accurate end-to-end network capacity detection method, [1] uses fuzzy reasoning to check whether there exists a wireless link in an end-to-end network path, and [12] presents a new accurate available bandwidth measurement tool based on a fisheye pattern of probing packets.

Using this set of tools, network connection types, link capacity, available bandwidth, jitter, and latency are measured and collected. These pieces of network awareness and system awareness information are then sent to the Priority Manager module inside the proxy, and create a user profile database, as shown in Fig. 2. It should be noticed that comparing to the pre-configured static user profile in other systems, the user profile data in our architecture is dynamically generated and continuously updated from the client to reflect the current conditions of the network and end system environment.

3.2. Application Priority QoS Specification

Priority is a basic concept for process scheduling in operating systems, where a process with higher

priority is scheduled more frequently and systems can dedicate more resources to this process. Here we extend this concept to specify different application QoS priorities. Consider that a user runs a downloading program and another online video streaming application at the same time on the same device. When a user assigns a higher priority to the video streaming application and a lower priority to the downloading application, the QoS of the video streaming application should be better preserved. The value of priorities represents the user QoS data, that is, the higher the priority value, the better application performance is expected by the end user. In this way, users can present the relative importance levels of applications and try to control (to some extend) the behavior of them via the QoS architecture.

The Program Priority Assignment program provides a simple way for an end-user to input the application priority through a graphic user interface (GUI). The GUI is flexible enough to let an end user specify the priority of a type of application (e.g. assigning the same priority to multiple instances of the Internet Explorers), priority of a type of network protocol (e.g. assigning the same priority to all applications that use the HTTP service), or distinct priority for each individual program. The system takes a default value for programs that do not have a priority value assigned by the user.

To relate an outbound network packet to its application, a few system APIs are used. EnumProc API enumerates current running process and reports their application process IDs. The lsof [8] is a tool that can list open files. Under Unix-type systems, an open file may be a stream or a network file (Internet socket, NFS file or UNIX domain socket). Through lsof, the OS knows whether a process has a socket open on a specified IP address/port or not.

In the current design, integer numbers (1 to 10) are used to indicate the priority. Since some users may assign all of his/her running applications to the highest priority, hoping that the proxy could provide more resources (e.g. computing, bandwidth etc) to this client. In order to be fair among multiple clients, each client has a maximum sum value of the priorities.

3.3. Inter-Application and Inter-client QoS Adaptation

Under the constraint of the QA-Proxy's resources, the essence of QoS is to provide the capability to differentiate between traffic or service types such that one or more classes of traffic or services can be treated differently than others [3]. Proxy-side QoS adaptation is performed by collaborations of Priority Manager, Traffic Distributor and Priority Queues, as shown in Fig. 2. After getting all these application priorities and client properties, Priority Manger decides the inter-application and inter-client (IAIC) priority of each packet from each client, and puts the incoming packet (i.e. outbound traffic) into a priority queue based on the IAIC priority calculation.

The information of clients provided by the Program Priority Assignment program includes running application priorities, application process IDs, and source/destination IP addresses and port numbers of the application. QA-Proxy assigns a unique ID for each application with its IP and network port. In case that a single application opens multiple network connections, multiple ports are assigned to the application and the QA-Proxy treats this application as multiple ones by assigning multiple IDs.

The priority of a client application running at a certain host, denoted here as H_P_i, is determined by multiple factors, such as the current network bandwidth, the mobility of the client, and the current host device capability, which are acquired by the Awareness Measurement tools located at each client. Then the inter-application and inter-client priority for each packet can be formulated as following:

$$P_{ij} = (O\_P_{ij} \times \alpha_1) * (H\_P_i \times \alpha_2)$$

where α_1 and α_2 are factors that put different weights to the inter-application and inter-client priority, P_{ij} represents an inter-application and inter-client priority value of the packet from the j^{th} application running at the i^{th} client, and O_P_{ij} represents the original priority value of the packet from the j^{th} application running at the i^{th} client. It is specified by the end user and sent from the client to the Priority Manger at the proxy.

In some situations that network administrators need more flexibility over the formulation or when the new inter-application and inter-client priority cannot be calculated, a priority look-up table can be used. Bandwidth, mobility, program priority, application throughput are inputs, and the inter-application and inter-client priority is the output (see Table 2). For example, a wireless client gets a higher priority than a wired client if other conditions are the same.

Table 2: An example priority look-up table

IAIC priority	Mobility	Bandwidth (bps)	Program Priority	Application Throughput
8	Wired	100M	10	10K bps
9	Wired	10M	10	10K bps
7	Wired	54K	8	1K bps
9	Wireless	10-19.2 K	8	10K bps

3.4. Priority Queues

The QA-Proxy receives outbound network traffic from its clients inside the intranet. A Traffic Distributor dispatches the outbound packets to one of the first-in-first-out (FIFO) queues. The number of queues is the same as the number of priorities in the system, e.g. 10. The Traffic Distributor extracts the source IP address and the port number from the packet, therefore it knows the process ID from which the packet is originated. And then it calculates or looks up the inter-application and inter-client priority (IAICP) of the packet, and dispatches the packet to the corresponding priority queue based on the IAICP. This process has been depicted as Fig. 3.

There are a fixed number of system threads in the Thread Pool. Each queue gets assigned a number of threads, calculated by the algorithm below, to process the data. Since all threads are scheduled by the OS evenly, the more threads for a queue the faster the queued data are processed. It is natural to conceive that more threads will be assigned to the highest priority queue. However, the thread assignment method is not purely driven by priority. It also takes into consideration the size of the total packets that have been stored in one queue. The thread assignment algorithm is formulated as follows:

$$\sum T_i = MAX\_THREAD\_NUM \qquad \frac{L_1}{T_1} < \frac{L_2}{T_2} < \frac{L_3}{T_3} < .... < \frac{L_k}{T_k}$$

If L_i=0 then T_i=0, and L_i is the average data length stored in the i^{th} queue during the recent period of time, e.g. 30 seconds; T_i is the number of threads assigned to i^{th} queue; k is the number of priorities in the system; and *MAX_THREAD_NUM* is the total number of threads used to process the priority queues. By satisfying the above conditions, a balance is established between the priority and the timeliness of processing the stored packets.

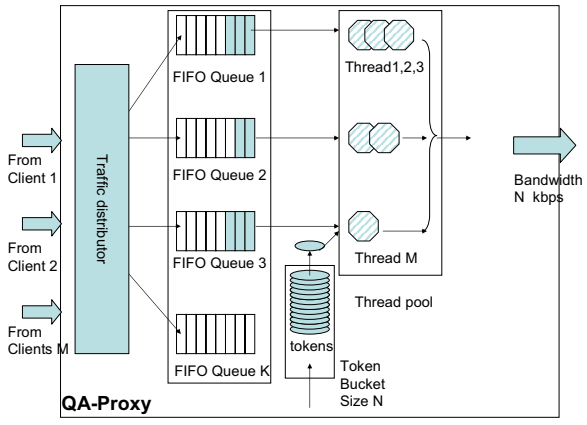

Figure 3: QA-Proxy components – sending outbound packets from clients to servers

The starvation problem is avoided by assigning at least one thread to each queue that is not empty. Since all threads are scheduled evenly, the lower priority queue is processed slower, but not suspended, if any data is in the queue. Until a priority queue is empty, all threads serving the queue will not be released.

The number of threads assigned to each queue depends on two factors: the priority of the queue and the amount of queued data. Lager number of threads associated with higher priority queues helps to empty the queues faster, if their queued data amounts are same. Dynamically reallocating more threads to a queue with growing amount of queued data helps to empty these queues more quickly, therefore it ensures that the service will be offered swiftly, without significantly affect TCP timeout mechanism. FIFO rule is used for each queue in order to preserve the original packet sequence from the application.

The dynamic assignment of the threads to the queues maximizes the global utility of the bandwidth from the QA-Proxy to the Internet. In fact, token bucket control mechanism [3] is used for each thread to maximize the utility and avoid congestions as well. The proxy puts tokens into the bucket in a consistent rate. The bucket can hold a maximum fixed number (*N*) of tokens, which correspond to the network interface bandwidth (*N* Kbps). The number of tokens that each thread can retrieve from the bucket is *N*/*MAX_THREAD_NUM*. If there is no token inside the bucket, the thread must wait.

Based on these rules, the QA-Proxy assigns more resources to applications with higher inter-application and inter-client priority values. Since all active threads share the bandwidth from QA-Proxy to the Internet, such applications will be served with more chances of getting data processed comparing to other applications with lower priorities. By balancing the resource distribution between high priority applications and low priority ones, the system performance expected by the users can be offered accordingly. Programs could be integrated in this QoS architecture transparently without source-code modifications.

Moreover, the resource availability such as the network bandwidth may change from time to time. Due of its position, the QA-Proxy is able to obtain the information of the available bandwidth between itself and data servers and the available bandwidth from itself to the end hosts. Data servers are servers in the intranet shown in Fig. 1, where different fidelities of the same data are stored. By adding a byte of additional information in the header of data requests, QA-Proxy could ask the data servers to adjust the data fidelity for adaptation to the variation of network conditions between them. In the same way, it could

change the amount of information in transmission by adding or discarding some contents of the packets to the client. The idea of making a tradeoff between data fidelities and the performance of data transmission has been studied in [4][10]. The proxy components when proxy sends inbound packets from servers back to clients are shown in Fig. 4. The architecture is similar to what has been shown in Fig. 3, except that there is one priority-queue array to each client. The token bucket is again used to control the network usage from the QA-Proxy to the individual client.

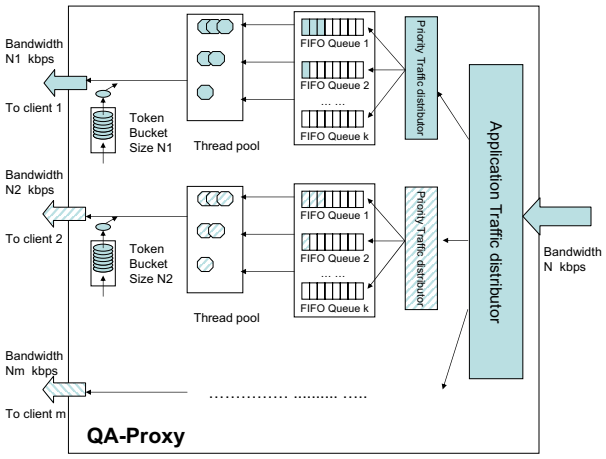

Figure 4. QA-Proxy components – sending inbound packets from servers to clients

4. Conclusions and Future Work

Most existing QoS-aware architectures and systems in heterogeneous network environments require applications to modify their source code to implement network-adaptation functionality. In this paper, we introduce a proxy-based QoS-aware architecture where legacy applications can be integrated transparently to adapt to the variation of system resource conditions, network variations, and user preferences by implementing QoS adaptation policies in a QoS-aware proxy in a three-tier client-server paradigm.

In order to provide the fairness among end users and applications, policies that drive inter-application and inter-client priority assignment should be fully adjustable considering different usage scenarios. Although we can use the priority look-up table in cases when the policies are too complex to be generated, we are still working on approaches to specifying the priorities more elegantly. The evaluation of the performance degradation in terms of packet latency due to the QA-Proxy is also expected. Moreover, a feedback such as a notification or recommendation of a good priority is necessary for applications that perform poorly because of a low priority value assigned. Finally we plan to deploy and test the proxy architecture in an educational setting.

This work is sponsored by National Science Foundation (NSF Award #0438300).

5. References

[1] L. Cheng and I. Marsic, Fuzzy Reasoning for Wireless Awareness, *International Journal of Wireless Information Networks*, Vol. 8, No. 1, pp. 15-26, 2001.

[2] L. Cheng and I. Marsic, Piecewise network awareness service for wireless/mobile pervasive computing, *Mobile Networks and Applications, ,* 7(4) pp. 269-278, 2002.

[3] P. Ferguson and G. Huston, Quality of Service: Delivering QoS on the Internet and in Corporate Networks, *John Wiely & Sons*, 1998.

[4] J. Bolliger and T. Gross, A framework-based approach to the development of network-aware applications, *IEEE Transactions on Software Engineering*, Vol. 24, No. 5, pp. 376-390, May 1998.

[5] M. Khedr and A. Karmouch, ACAN - ad hoc context aware network, *CCECE'02*, Winnipeg, Canada, May 2002.

[6] G. Welling and B.R. Badrinath, An architecture for exporting environment awareness to mobile computing applications, *IEEE Transactions on Software Engineering*, Vol. 24, No. 5, pp. 391-400, May 1998.

[7] K. Nahrstedt, D. Xu, D. Wichadakul, and B. Li, QoS-aware middleware for ubiquitous and heterogeneous environments, *IEEE Communications Magazine*, Vol. 39, Issue 11, pp. 140-148, November 2001.

[8] lsof tool: online at http://www.cert.org/security-improvement/implementations/i042.05.html

[9] B. Li, D. Xu, and K. Nahrstedt, An integrated runtime QoS-aware middleware framework for distributed multimedia applications, *Multimedia Systems*, Vol. 8, pp. 420–430, 2002.

[10] B.D. Noble and M. Satyanarayanan, Experience with adaptive mobile applications in Odyssey, *Mobile Networks and Applications,* Vol. 4, 1999.

[11] R. Wolski, N.T. Spring, J. Hayes, The network weather service: a distributed resource performance forcasting service for metacomputing, *Journal of Future Generation Computing Systems*, 15(5-6), pp. 757-768, Oct. 1999.

[12] Q. Wang and L. Cheng, FEAT: improving accuracy in end-to-end available bandwidth estimation, submitted to *Passive and Active Measurement*, 2005.

[13] X. Gu, D. Wichadakul, and K. Nahrstedt, Visual QoS programming environment for ubiquitous multimedia services, in *Proc. of IEEE International Conference on Multimedia and Expo 2001*, Tokyo, Japan, August 2001.

[14] D. Hoang, D. Reschke, W. Horn, Adaptive quality of service management using QoS proxy and user feedback for wireless links, *in Proceedings of the International Workshop on Innovative Internet Computing Systems* (IICS 2001), pp. 31- 40, Ilmenau, Germany, June 2001.