Usage Based Service Differentiation for End-to-End Quality of Service Management*

Michael J. Katchabaw, Hanan L. Lutfiyya, and Michael A. Bauer
Department of Computer Science
The University of Western Ontario
London, Ontario, Canada N6A 5B7

Abstract

It is often difficult to ensure that every application in a computing environment receives the level of quality of service required by their users. In such cases, the demand for computing resources to do so simply exceeds the limited supply that is available. To ensure that at least some user applications meet quality of service requirements, service differentiation is one approach growing in popularity. In this approach, preferential service is given to selected applications, while others deemed less important suffer in comparison. Most work in this area bases service differentiation decisions on static information about the applications, such as the name and type of application, the owner of the application, execution time, and the host on which the application was executed.

In this paper, we discuss a new approach to service differentiation that takes into consideration dynamic application usage information in service differentiation decisions. In doing so, we can make better or fairer service differentiation decisions that allow more users to enjoy higher levels of quality of service. This is accomplished by ensuring preferential service is given to applications whose users can actually benefit from the improved service, as opposed to those applications that are essentially ignored or in a state that renders them unusable.

[**Keywords:** Behavioural user hints, service differentiation, end-to-end quality of service management]

1 Introduction

At the heart of the quality of service management problem is resource management. If we had an infinite supply of every computing resource, we would be able to satisfy the quality of service requirements of all user applications simultaneously. Since this is not the case, effective resource management is necessary. Traditionally, resource management in computing environments has used a uniform service discipline, in that all applications have equal access to computing resources to satisfy user requirements. However, in times of overload, when resources become scarce, it is possible for no applications to satisfy user quality of service requirements, which is highly undesirable.

As an alternative, a new service discipline has been proposed—differentiated services. In this approach, computing resources are managed to provide preferential service to certain applications, at the cost of reduced service to other applications. With differentiated services, the computing environment can still ensure some applications satisfy user quality requirements, even in conditions of extreme overload. Early work in service differentiation focussed on the management of network resources [6], but new interest has been given to end-to-end approaches that provide service differentiation at the network, middleware, operating system and application levels.

The effectiveness of a service differentiation solution largely depends on the decisions made by the service differentiation system in determining which applications receive preferential treatment and which ones suffer as a result, as well as the methods used to provide this preference. Unfortunately, most approaches to service differentiation use relatively static information in their decision processes [21]. This includes the application name and type, the owner of the application, the host the application was executed on, the time the application was executed, and so on. Typically, service differentiation decisions do not take into account the actual behaviour of users and their applications as these applications execute. Such information, however, would be quite useful in making better or fairer service differentiation decisions that allow more users to enjoy higher levels of quality of service.

As an illustration, we conducted a study of computing environment usage by faculty, staff, and grad-

*An extended version of this paper can be found as [12].

Proceedings of the 2003 Workshop on End-to-End Service Differentiation,
Phoenix, Arizona, April 2003

uate students in our department [11], which revealed that users frequently ignore executing applications for extended periods of time, which can result in significant wastage of scarce computing resources. These scenarios result in resources being wasted and spread thin over some applications unnecessarily, and quality requirements for other applications not being satisfied. With service differentiation applied, this is even more tragic when ignored applications are also being favoured by the service differentiation process. In such a case, applications suffering without preferential treatment are doing so when the preferred service is not being enjoyed and put to good use.

In this paper, we present a usage based approach to service differentiation in end-to-end quality of service management. In addition to a variety of static information, our approach also uses dynamic behavioural information in making service differentiation decisions. By collecting and tracking application visibility, keyboard and mouse focus, and other generic status information, we can determine which applications are the focus of user attention and activity and guide service differentiation decisions accordingly. In doing so, we can ensure that preferential service is being put to good use, and not wasted on applications that are not usable or are being ignored. This approach is used in conjunction with our previous work on policy based end-to-end quality of service management [11] to provide a comprehensive and powerful service differentiation mechanism.

The remainder of this paper is structured as follows. Section 2 examines related work in this area. Section 3 presents our approach to service differentiation and an architectural framework for quality of service management. Section 4 discusses a prototype implementation of this framework with policies to provide service differentiation. Section 5 presents results from a variety of experiments conducted using the prototype system. Section 6 concludes this paper with a summary and discussion of further avenues for continued research in the future.

2 Related Work

In recent years, there has been considerable work in the area of quality of service management. A lengthy survey of such work can be found in [11]. In this section, we will focus on the two bodies of research in this area closest to this work—service differentiation and behavioural quality of service management.

2.1 Service Differentiation

Much of the work in service differentiation for improving quality of service has focussed on the network layer. The canonical work in this area is DiffServ [6]. Further work has been done in [4, 8, 17], and elsewhere. Network level service differentiation on its own, however, is not enough. An end-to-end solution is required encompassing the network, middleware, operating system and application levels. Consequently, the above work on its own is not sufficient. Simply put, effective management of all computing resources is required for satisfactory quality of service [7]. However, many of the concepts and techniques used in network level service differentiation are quite useful and applicable to other layers as well, and should not be ignored or forgotten.

Relatively little attention has been focussed on service differentiation above the network level. Much of this work has been in the context of web services and electronic commerce applications. Such work includes [2, 5, 9, 19, 22] and others. While this work introduces several promising techniques for service differentiation, they are targeted towards a specific application domain, and it is unclear how transferable they are to other types of applications. Furthermore, virtually all of the above cases only considered static information in service differentiation decisions. As noted earlier, however, it is desirable to include more behavioural information in these decision making processes.

2.2 Behavioural Quality of Service Management

Using user behaviour in quality of service management is a relatively new concept. This issue was raised in [1], but unfortunately no design or implementation work was ever completed. [18] also examines this issue, providing an environment that fed into the quality of service management process when users interacted with it. However, this was only done in the context of their own proprietary 3D landscape environment, and not generally available production environments. The SCUBA (Scalable ConsensUs-based Bandwidth Allocation) protocol [3] is aimed at allowing multimedia sources to account for receiver interest in adjusting transmission rates. While this work is interesting, it is geared towards only a particular class of applications, and does not adjust resources allocated or consumed; instead only modifying application behaviour. Additional research into multimedia applications such as [14, 15] and others has yielded adaptive applications capable of responding to user feedback (for example, by not transmitting audio streams when audio is muted by the user). Unfortunately, much of this work is domain-specific and limited in scope, quite often only to reducing work done in response to user feedback, without deallocating and descheduling resources, which can lead to further problems.

2.3 Summary

While some work has been done towards integrating application usage information into the quality of service management process, no work to date has examined doing so in the context of service differentiation. Concepts from each area must be combined to provide service differentiation mechanisms that take this usage information into account. This is precisely what we have done, in combining our own earlier work in end-to-end service differentiation [16] and behavioural quality of service management [13]. Our approach to doing so is discussed in the next section.

3 Our Approach

In this section, we examine the approach taken in our work for usage based service differentiation in quality of service management. We begin with an overview of our approach, followed by a discussion of a management architecture that realizes this approach.

3.1 Overview

There are three key elements to our approach to usage based service differentiation:

Policy Based Service Differentiation: Policy based management is an approach to management in which the requirements used to govern management decisions are logically separated from the underlying mechanisms used to actually carry out management activities. In this approach, the formalism used to capture these requirements is policies. In the literature, policies are generally defined as if-then rules: if the appropriate conditions are met, then a corresponding set of actions are carried out as a result [20]. In policy based service differentiation, policies specify the conditions under which service differentiation should apply, rules dictating which applications should receive preferential quality of service and which should not, and the actions required to actually differentiate service delivery.

Dynamic End-to-End Resource Management: As mentioned earlier, effective quality of service management heavily depends upon an effective underlying resource management scheme. The basic resource management procedure taken in our work is to iteratively allocate resources to application processes, measure the change in delivered quality as a result, and use these differences as feedback in determining further adjustments. This is repeated until quality of service requirements are satisfied. An iterative approach to quality of service management makes sense, since attempting to move delivered quality to requirements in a single step is too expensive computationally, and moving forward in iterations allows management to cope with changes resulting from the dynamic nature of the problem.

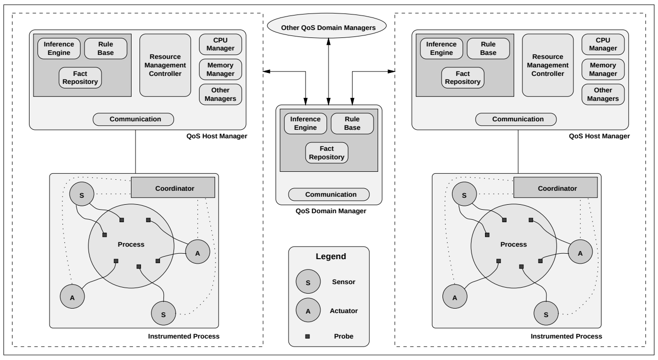

Figure 1: Service Differentiation Architecture

Behavioural User Hints: Since measuring user behaviour directly is a difficult process, we instead track hints of user behaviour from their interactions with their applications. These user hints can be used to infer user interest and activity that can be used to help guide management decisions. Examples of user hints include change of mouse and keyboard focus amongst applications, changes in application state (minimized or maximized), changes in application viewability (covered or uncovered), application idle time, the presence or absence of screen savers or locking mechanisms, and so on.

Hints of user interest and activity are fed into the policy based service differentiation mechanism, and are checked as additional conditions on policy rules. For example, we could have a policy state that if an application does not have user focus or if it is unviewable or unusable, that it should not receive preferential quality of service. In such cases, we could have these usage based service differentiation policies supersede or override other service differentiation policies in guiding service differentiation decisions.

3.2 Usage Based Service Differentiation Architecture

In this section, we discuss a software architecture developed to support usage based service differentiation, based on our earlier work in quality of service resource management [11]. This architecture is shown in Figure 1, and is discussed in the sections below.

3.2.1 Application Instrumentation

Application instrumentation allows for optimal monitoring and control of application behaviour to facilitate management on a consistent and uniform basis.

Instrumented Process

An instrumented process is an application process with instrumentation code embedded at strategic locations to facilitate management. It is this instrumentation code that enables the monitoring and manipulation of the application process by the quality of service resource management system.

Coordinator

The coordinator facilitates interactions between an instrumented process and the quality of service management system. Its role includes message routing, managing other instrumentation components, and handling initialization and termination activities within the process. All knowledge of the management system is confined to this component, effectively hiding it from the remaining instrumentation components.

Sensors

A sensor encapsulates an attribute characterizing some element of application behaviour. Sensors get their input data from probes inserted at strategic points in the application code or by reading other sensors. Sensors provide their information to the coordinator in the form of periodic reports, alarm reports (when exceptional or critical circumstances arise), or in response to explicit requests. Sensors are dynamic components—they can be enabled or disabled, reporting intervals can be adjusted, thresholds can be modified, and sensor processing algorithms can be changed, all at run-time while the application process is executing. Sensors not only collect quality of service measurements to determine if application quality of service requirements are being met, but sensors are also used to collect information used in the service differentiation process. This includes both static information about application processes (the application name, type, owner, time of execution, and host of execution), as well as behavioural user hints used to guide the service differentiation process.

Actuators

Actuators are used to encapsulate functions that can exert control over an instrumented process to change its operation or behaviour. Actuators, like sensors, function through the use of probes or other actuators, and possess similar dynamic properties. Actuators are useful to support quality of service negotiation, adaptation, service differentiation, and a variety of other quality of service functions.

Probes

A probe is an interface between process code and instrumentation sensors and actuators. Typically, probes take the form of method invocations or function calls to facilitate interactions with sensors and actuators. Probes allow sensors to collect attribute information and allow actuators to exert control over process behaviour. Each probe is specific to a particular sensor or actuator. Probes are the only instrumentation component that require modifications of the original process source code—all other components need only reside in the same address space as the process, and do not require such modifications.

3.2.2 Quality of Service Managers

The basic idea behind our approach is to provide a quality of service management process local to each host, responsible for the quality of service delivered by applications executing on the host. Application process coordinators communicate with these QoS Host Managers to report violations of quality of service requirements and to return monitored data upon request from its QoS Host Manager. Coordinators may also receive requests from its QoS Host Manager to manipulate application behaviour or tune instrumentation behaviour. With applications distributed across multiple hosts in a computing environment, problems in satisfying quality of service requirements can span these hosts. To facilitate the quality of service management process in such cases, a separate domain manager process coordinates the activity of the individual local host managers. This QoS Domain Manager is responsible for overseeing the quality of service provided by a collection of hosts; it resides on one of these hosts. It is responsible for making decisions, leaving implementing these decisions up to the appropriate QoS Host Managers. If problems span multiple such collections, additional coordination is necessary.

To carry out its tasks, a quality of service manager is composed of a variety of components, including a fact base, a rule base, an inference engine, a resource management controller, and a collection of resource managers. These components are discussed below.

Communication

The quality of service manager communication module is used to facilitate communication between the manager and other processes in the computing environment. Local inter-process communication mechanisms are used to communicate with processes since they reside on the same host as the host manager, providing efficient and reliable communication. Remote communication mechanisms are used to communicate with the domain manager since it may reside on any host in the domain.

Fact Base

The fact base is used to contain information known by a quality of service manager for diagnosis and planning purposes. The information contained in the fact base includes the following: information on managed processes; data provided through quality of service violation reports, when quality of service no longer meets requirements; behavioural user hints used to guide service differentiation decisions; and additional data queried from the operating system or other application processes as necessary.

Rule Base

The rule base contains a set of rules used in diagnosing and planning to determine corrective actions to quality of service problems. Each rule contains a set of conditions on when to apply the rule (which facts must be present) and the actions to take when the rule is applied. Rules are developed with some type of expert knowledge. Policies required by a QoS Host or Domain Manager are mapped to one or more rules loaded into the rule base. To support usage based service differentiation, a variety of rules specific to this task are loaded into the rule base. These rules are used to determine when service differentiation should be applied, and how it should be applied.

Inference Engine

Inferencing is a recursive process whereby the inference engine determines the set of rules applicable based on the facts in the fact base, and applies them to formulate new facts for the fact base or determine corrective actions. The inference engine continues to apply rules as long as there are rules that are applicable. When done, a set of corrective actions has been accumulated and is ready to fire. The manipulations required are forwarded to the resource management controller, if resource adjustments are necessary, or to application processes, for other adaptations.

Resource Management Controller

The resource management controller module oversees quality of service resource management for the local host. It makes general resource management decisions, leaving the details to individual resource managers.

Resource Managers

Resource managers are used to provide control over the management of local resources. This includes the allocation, scheduling, and accounting for these resources. Resource managers typically manage an individual resource, but they may also manage multiple resources. Managers encapsulate all of the details of working with the actual resources; consequently, the resource management controller does not need to concern itself with these details. This split of concern makes it easy to develop resource management algorithms independently of the mechanisms used to allocate them, and also allows new resource management mechanisms to be developed and put in place without having to substantially re-engineer the algorithmic logic.

4 Prototype Implementation

To evaluate the approach, we have developed a prototype system based on the architecture presented in the previous section. This prototype has been implemented for the Solaris 2.6 environment.

4.1 Instrumentation Implementation

Instrumentation is made available to application developers through a C++ class library; a C interface to the class library is also available. Instrumentation components have been developed to support a wide variety of applications. User hint sensors have been provided to detect and handle generic behavioural user hints in an X Windows environment (application covering and uncovering, minimizing and maximizing, screen saver and lock activation, excessive idle time, keyboard and mouse focus, and so on) as well as application-specific hints in video applications (audio muting, video stoppage or pausing, and so on). Further implementation details of our instrumentation and code examples can be found in [11].

4.2 QoS Manager Implementation

Both the QoS Host and QoS Domain Managers are multithreaded daemon processes implemented in C and C++. As daemon processes, these managers are servers that are continuously running in the background; they are executed by the system when it boots, and not executed explicitly by users. Both daemons spawn children processes to handle multiple concurrent policy violations and various other requests as necessary.

The QoS Host Manager adjusts allocations dynamically through a collection of resource managers that each manage a single system resource. To date, we have resource managers capable of adjusting CPU and memory; network resource adjustment is currently being added to our prototype [4]. There is one QoS Domain Manager in the local network that receives reports from QoS Host Managers. It has rules in its rule base that can distinguish between a remote machine problem and a network problem (this usually requires a query of other QoS Host Managers) and can tell a QoS Host Manager on a remote machine to adjust resource allocations where appropriate.

The fact base, rule base, and inference engine components of QoS Host and QoS Domain Managers are implemented through the use of the CLIPS (C Language Integrated Production System) inference engine [10]. CLIPS uses first order predicate logic to implement an expert system. Because it is easily extended, embeddable in applications, freely available in source form, and supports dynamic loading and changing of facts and rules, CLIPS was a good candidate for this implementation. We make use of a slightly stripped and optimized version of CLIPS for greater performance and efficiency. Further implementation details of the managers can again be found in [11].

4.3 Service Differentiation Policies

The logic behind service differentiation decisions is formally captured as a collection of policies. These policies are implemented as rules used by the CLIPS engine to determine when service differentiation should be used, which applications should be given preferential treatment and which application should not, and how service differentiation should take effect. Depending on the facts for an application in the fact base, the inference engine will use different rules to apply service differentiation accordingly.

To date, rules have been implemented to provide differentiated services based on the application name, application type, owner of the application, time of execution, the host on which the application is executed, and current application conditions, as determined by behavioural user hints present in the fact base. As mentioned earlier, these hints would allow rules to trigger on various aspects of application state, including whether the application is idle or not, covered or uncovered, minimized or maximized, or has keyboard or mouse focus. This allows service differentiation decisions to be based not only on static information, but also on usage information. In the future, additional run-time information could be used in the decision process, including the amount of resources each application consumes, the estimated finishing time of the application, and so on. Currently, applications not selected for preferential service are typically either reverted back to best effort service, or have their resource allocations scaled back to free resources for favoured applications. Other possibilities would be to suspend or terminate the applications, or ask the applications to renegotiate quality of service requirements or adapt their behaviour to meet current requirements using fewer resources. Additional implementation details and examples of service differentiation policies can again be found in [11].

5 Experimental Results and Experience

For experimentation, we chose to use a distributed client-server video-on-demand application. In response to a request from a client, the server streams a given MPEG video over a TCP connection back to the client, which decodes the video and presents it to the user. The application workload assigned was a 10 minute 320×240 8-bit video of footage recorded by a camera at our department's I/O counter and stored in a local file. The quality of service requirements in place dictated that this workload be delivered at a mean frame rate of 24 frames per second, plus or minus 2 frames per second (yielding an acceptable quality range from 22 to 26 frames per second).

Three machines from our laboratory environment were used in experimentation: `farquarson`, a Sun Microsystems UltraSparc 1 workstation with a 140 Mhz processor, 64 megabytes of physical memory, and 300 megabytes of virtual memory; `strawberry`, a Sun Microsystems UltraSparc 5 workstation with a 269 Mhz processor, 128 megabytes of physical memory, and 480 megabytes of virtual memory; and `vanilla`, a Sun Microsystems UltraSparc 5 workstation with a 269 Mhz processor, 256 megabytes of physical memory, and 760 megabytes of virtual memory. All three machines use Solaris 2.6 as their operating system.

Initial experimentation, discussed in detail in [12], yielded the following results. First, we found that a single instance of the application could deliver acceptable quality with no problems. Second, we found that two instances of the application, with servers on the same host and clients on different hosts, completely missed quality requirements. Third, we found that with management in place using a uniform service policy to manage both applications, quality was more predictable and greatly improved, but neither application could obtain sufficient resources to meet requirements. Fourth, we found that with service differentiation in place in the above scenario, instead of a uniform ser-

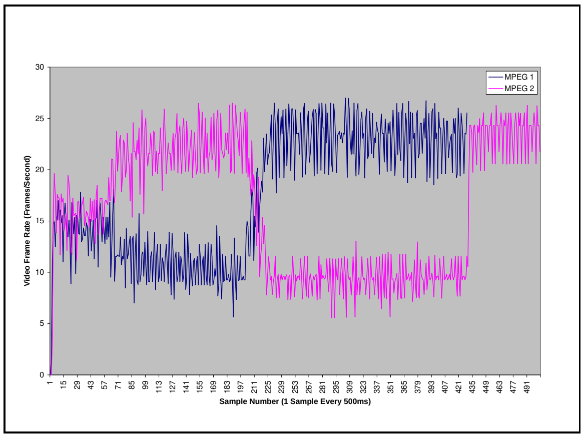

Figure 2: Usage Based Differentiated Services Results

vice discipline, we could effectively favour one application over the other. We observed in this case, however, that since user behaviour was being ignored in management decisions, changes in application visibility had no impact on resources allocated or quality delivered, which could be quite unfair to the application remaining visible.

To demonstrate usage based service differentiation, we used the same service differentiation policy as mentioned above to favour the first application launched over the second instance, with the second instance still aiming to deliver quality at a level of 50% of its requirement. This time, however, behavioural user hints were collected, and additional service differentiation rules were added. These new rules dictate that if an application is minimized or completely covered by another application, then it cannot receive preferential service by our other service differentiation policies. In effect, it allows usage information to override other discriminators used in service differentiation decisions.

Figure 2 presents a representative run of this experiment under usage based service differentiation. The first instance of the video-on demand application starts, followed very shortly by the second instance. The second instance completely obscures the first instance when it starts. When the two applications exhaust resources available for allocation, service differentiation is enacted. Since the first instance of the application is obscured, our usage based service differentiation policy indicates that the second instance should be favoured in service differentiation. The resulting changes in resource allocation, allow the second instance to meet its quality requirements, while the first still manages to meet its 50% requirement. When the first instance is uncovered, the original service differentiation policy guides resources to this instance, allowing it to meet requirements while the second instance suffers in its place. In other words, this approach behaves precisely as it should, ensuring quality of service is delivered according to both static information and application usage as well.

6 Concluding Remarks

The work we describe here is part of ongoing work on addressing issues in end-to-end quality of service management. This paper focusses on the integration of usage information into the service differentiation process to provide better or fairer service differentiation decisions that allow more users to enjoy higher levels of quality of service. We developed a general approach and architecture for facilitating this procedure. Using a prototype system based on this approach, we have been able to carry out a number of experiments that demonstrate that usage based service differentiation is possible, and can produce very useful results. This approach allows us to ensure that preferential service is given to applications and users that can benefit from it, and not wasted on applications that are being ignored or are of no use to anyone.

We have identified several avenues for continuing and future work in this area. We need to investigate other usage information that can be used in the service differentiation process to improve service differentiation decisions. We need to examine issues in policy specification to allow users and administrators to develop service differentiation policies with less effort. We need to complete our work integrating network resource management into our management system, as well as facilities for I/O and disk resource management. We need to continue porting efforts to other platforms, including the Microsoft Windows family, other variants of the Unix environment, and Java.

Through this work, we have gained valuable insight into the service differentiation problem. With this experience, we can further pursue solutions that completely address the variety of quality of service problems that occur in today's computing environments.

References

[1] M. Alfano. Design and Implementation of a Cooperative Multimedia Environment with QoS Control. *Computer Communications (21), Elsevier Science*, Fall 1997.

[2] J. Almeida, M. Dabu, A. Manikutty, and P. Cao. Providing Differentiated Levels of Service in Web Content Hosting. *Proceedings of the 1998 SIG-*

METRICS Workshop on Internet Server Performance, Madison, Wisconsin, March 1998.

[3] E. Amir, S. McCanne, and R. Katz. Receiver-driven Bandwidth Adaptation for Light-weight Sessions. *Proceedings of the Fifth ACM International Conference on Multimedia*, Seattle, Washington, November 1997.

[4] M. Bauer and H. Akhand. Managing Quality-of-Service in Internet Applications Using Differentiated Services. *Journal of Network and Systems Managment, Volume 10, Number 1*, March 2002.

[5] N. Bhatti and R. Friedrich. Web Server Support for Tiered Services. *IEEE Network*, September/October 1999.

[6] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. *IETF RFC 2475*, December 1998.

[7] H. Chu and K. Nahrstedt. A Soft Real Time Scheduling Server in UNIX Operating System. *European Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services*, Darmstadt, Germany, September 1997.

[8] C. Dovrolis, D. Stiliadis, and P. Ramanathan. Proportional Differentiated Services: Delay Differentiation and Packet Scheduling. *Proceedings of IEEE/ACM Transactions in Networking*, February 2002.

[9] L. Eggert and J. Heidemann. Application-Level Differentiated Services for Web Servers. *World Wide Web Journal Volume 2, Number 3*, August 1999.

[10] J. Giarratano and G. Riley. *Expert Systems: Principles and Programming.* PWS Publishing Company, 1998.

[11] M. J. Katchabaw. *Quality of Service Resource Management.* PhD thesis, The University of Western Ontario, June 2002.

[12] M. J. Katchabaw, H. L. Lutfiyya, and M. A. Bauer. Usage Based Service Differentiation for End-to-End Quality of Service Management. *Technical Report #587, Department of Computer Science, The University of Western Ontario*, January 2003.

[13] M. J. Katchabaw, H. L. Lutfiyya, and M. A. Bauer. Using User Hints to Guide Resource Management for Quality of Service. *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, Nevada, June 1999.

[14] S. McCanne and V. Jacobson. LBNL Audio Conferencing Tool (vat). *On-line Documentation from http://www-nrg.ee.lbl.gov/vat*, December 1996.

[15] S. McCanne and V. Jacobson. vic: A Flexible Framework for Packet Video. *Proceedings of ACM Multimedia '95*, November 1995.

[16] G. P. Molenkamp, M. J. Katchabaw, H. L. Lutfiyya, and M. A. Bauer. Distributed Resource Management to Support Distributed Application-Specific Quality of Service. *Proceedings of the Fourth IFIP/IEEE International Conference on the Management of Multimedia Netowkrs and Services*, Chicago, Illinois, October 2001.

[17] A. Odlyzko. Paris Metro Pricing: The Minimalist Differentiated Services Solution. *Proceedings of the Seventh IEEE/IFIP International Workshop on Quality of Service*, London, England, June 1999.

[18] M. Ott, G. Michelitsch, D. Reininger, and G. Welling. An Architecture for Adaptive QoS and its Application to Multimedia Systems Design. *Computer Communications Special Issue on Guiding Quality of Service into Distributed Systems*, August 1997.

[19] R. Pandey, J. Barnes, and R. Olsson. Supporting Quality of Service in HTTP Servers. *Proceedings of the 17th SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, June 1998.

[20] Startdust.com. Introduction to qos policies. Technical report, Stardust.com, Inc., July 1999.

[21] N. Vasiliou. Overview of Internet QoS and Web Server QoS. *Reading Course Report, Department of Computer Science, The University of Western Ontario*, April 2000.

[22] N. Vasiliou and H. Lutfiyya. Managing a Differentiated Quality of Service in a World Wide Web Server. *Integrated Network Management Volume VII*, May 2001.