More on 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
E-Mail: {katchab,hanan,bauer}@csd.uwo.ca

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.

To this end, we have formulated an approach and developed an end-to-end quality of service management system capable of usage based service differentiation. In this paper, we present our general strategy, the design of a solution realizing this approach, and a preliminary prototype implementation based on this architecture. We describe experimentation and experience to date and evaluate our work and its effectiveness based on these preliminary results. Finally, we conclude with a summary of our work and outline our plans to evolve it in the future.

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

1 Introduction

The quality of service delivered by an application can best be defined as the level of satisfaction a user derives from the use of the application—whether it meets, falls short of, or exceeds the user's expectations [41]. User satisfaction is impacted by factors that include performance, reliability, and availability. As an example of a quality of service requirement, a user may require that a video-on-demand application display its video at a frame rate of 24 frames per second, plus or minus two frames, in order to be satisfied with the application.

In today's computing environments, meeting the quality of service requirements of users is becoming increasingly difficult. The typical environment is widely distributed, consisting of hundreds or thousands of heterogeneous computing nodes, and is growing in both size and diversity at a rapid pace. Applications in these environments are becoming larger, more complicated, and multimedia in nature, yet they must still co-exist with more traditional applications for transaction processing, data processing, and software development. Increasingly, these settings are playing mission critical roles in the enterprise, and their quality of service is essential. Indeed, users are demanding levels of performance, reliability, and availability comparable to those on traditional centralized systems, where quality of service problems are smaller, better contained, more readily identified, and easier to resolve. Consequently, meeting the quality of service requirements of users is becoming a considerable challenge for today's administrators.

Quality of service management is the set of activities required to ensure that user quality of service requirements are met. At the heart of quality of service management 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 manage-

Technical Report #608, Department of Computer Science, The University of Western Ontario
London, Canada, November 2003

ment 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.

An alternative service discipline has been proposed to overcome the weaknesses of uniform services—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 [7], but new interest has been given to end-to-end approaches that provide service differentiation at the network, middleware, operating system and application levels (as evidenced by the appearance of workshops and conferences dedicated to the subject, such as the Workshop on End-to-End Service Differentiation held in conjunction with the IEEE International Performance Computing and Communications Conference).

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 [39]. 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.

To illustrate the importance of considering user behaviour in quality of service management decisions, we conducted a study of computing environment usage by faculty, staff, and graduate students in our department [25]. This study revealed that users frequently ignore executing applications for extended periods of time, which can result in significant wastage of scarce computing resources. Ignored applications were either minimized or covered by applications that were the focus of user attention, or covered by screen savers or locks that activated due to prolonged periods of inactivity. At any given time, with no screen saver or lock active, only 22% of the applications executing on average were completely visible and usable to the user. In the end, the average application spends only about 40% of its life completely visible and usable; the rest is spent either minimized or covered in some way. In this 60% of its life, however, the application still consumes resources, even though the application is essentially useless and is not providing meaningful service to users. Other studies and evaluations [4, 13, 22] echo these results. 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. Service differentiation must take usage and behaviour into consideration to avoid such undesirable consequences.

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 application usage information and patterns of user behaviour 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, thereby making quality of service management more effective and efficient. This approach is used in conjunction with our previous work on policy based end-to-end quality of service management [27, 30, 31] 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 [23]. 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 [7]. In this approach, packets are classified and tagged at network ingress points; these tags are then examined at each router to determine how the packets should be handled according to a variety of service models [17, 21]. Further work has been done in [12] to integrate DiffServ with Multi-Protocol Label Switching for additional flexibility and quality of service delivery options. Paris Metro Pricing [33] is a simple approach to network differentiated service using a pricing mechanism to provide congestion control and assign packets to traffic classes for later differentiation. Another approach to network differentiated services is the proportional differentiation model [10]. In this approach, the actual quality provided by each class of packet varies with load, but the ratio of quality between classes remains fixed. The work in [5] takes a slightly different approach, examining relative proportional differentiated services, and using feedback to dynamically adjust quality levels.

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 [8]. 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. The work [2] investigates service differentiation amongst several web sites co-located at the same server through priority based scheduling providing low and high priority classes. The work in [35] provides differentiation on a per-page basis, allowing a site to customize how the server prioritizes page requests and allocates server resources. The work in [11] uses a variety of server-side application-level mechanisms to provide two levels of quality of service, regular and low priority. Through a variety of classification, admission control, and scheduling techniques, the work in [6] provides several quality of service levels for different classes of users. The work in [40] classifies web requests by client identifiers and other information, and then employs a variety of scheduling algorithms to provide different levels of quality of service that can be tuned dynamically at run time.

While the above 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. [34] 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 [28, 29, 38, 42] 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.

Microsoft Research is studying similar issues with the design of its next generation operating systems and applications based on its new ".NET" strategy, aiming for "attention sensitive user interfaces". The Lumiere project [19], for instance, uses cameras and microphones, in addition to other user feedback, to determine the goals and needs of users. Currently, this data is being used primarily for alerting and communicating with the user [20]; however, it could very easily be applied for adjusting resource allocations to user applications to improve quality of service. Other work in [18] aims to use idle time when users are not fully using resources for precomputation and analysis, which could be used in making more informed resource management decisions in the future.

Along similar lines is work in analysing patterns of user actions for a variety of purposes, including adaptive resource scheduling. In theory, by analysing what a user does, and comparing that to a known history of the particular user or users in general, one can predict what a user is about to do, and provide resources in advance. Unfortunately, research in this area has yet to reach this point.

The work in [9, 32, 26] examines extracting behavioural patterns from user command sequences and other information in Unix environments, but only use these patterns to preemptively launch new commands, without special resource considerations for quality of service. [14, 15] examines how to adapt scheduling based on user activity, but only in the context of particular industrial settings, and not in terms of general computing environments.

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 research area need to 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 [30] and behavioural quality of service management [24]. 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 end-to-end quality of service management. We begin with an overview of the key characteristics our approach, followed by a discussion of a management architecture that realizes this approach. After examining the details of this architecture, we examine how the various components function and interact with each other to provide usage based service differentiation.

3.1 Key Characteristics of Our Approach

Usage based service differentiation entails providing mechanisms that allow information on application usage and user behaviour to be used in the service differentiation process for quality of service management. For example, if we detect that a user is essentially ignoring a particular application, there is little point in providing this application preferential treatment in resource allocation decisions, especially when such resources are scarce.

There are three key characteristics of our approach to usage based service differentiation. Each characteristic is described below.

Dynamic End-to-End Resource Management: A dynamic resource management scheme is important to usage based service differentiation. With resources managed dynamically, we can enact service differentiation only when doing so is necessary; for example, when there are not sufficient resources to satisfy each application's quality of service requirements at the same time. Dynamic resource management also provides for a wider variety of differentiation options, which will be discussed later in Section 4. Finally, dynamic resource management allows us to change resource allocations and how service differentiation is applied in accordance with changes in the way applications are being used. For example, when user focus shifts, or an application becomes unviewable or unusable, we can shift resources accordingly and apply service differentiation to ensure applications that are the focus of activity or are usable receive preferential quality of service.

The basic resource management procedure taken in this work is to iteratively allocate or deallocate resources to or from 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 resource allocations makes sense, since attempting to move delivered quality to requirements in a single step is too expensive computationally. Furthermore, moving forward in iterations allows management to cope with changes resulting from the dynamic nature of the problem, including changes in resource availability, application workloads, and user requirements.

Policy Based Service Differentiation: In this approach, the formalism used to capture quality of service 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 [37]. For example, an expectation policy for a video-on-demand application could specify a condition that if the frame rate of delivered video falls below 22 frames per second, actions would be taken to notify the quality of service management system of the problem. These policies can be specified by application users, developers or administrators.

In policy based management, such policies are used to drive the management process and govern management decisions. Based on these decisions, underlying mechanisms are used to actually carry out management activities. From an architectural point of view, the use of policies decouples the logic of what management is to accomplish from the mechanisms to achieve these accomplishments, providing a flexible and dynamic management solution. In policy based service differentiation, policies specify when service differentiation should apply, which applications should receive preferential treatment and which should not, the form that preferential treatment should take, and the actions required to actually impose service differentiation.

Behavioural User Hints: Since measuring user behaviour directly is a difficult process, we instead monitor 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 inserted 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.

In the following section, we present a management architecture that embodies these characteristics and integrates them into a service differentiation solution.

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 [23]. This architecture is shown in Figure 1, and is discussed in detail 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. While some monitoring can be made by observing external application behaviour and rudimentary manipulations can be achieved through operating system interactions, work in this area has found that these approaches are limiting in both accuracy and flexibility in the kinds of metrics and control available. Since these limitations are unacceptable for quality of service management, application instrumentation is the best option.

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.

Suppose, for example, that user quality of service requirements for a client-server video-on-demand application dictate that the video-on-demand client should display streamed video to the user at a rate of 22 to 26 frames per second, and that if this requirement is not met, a notification of this violation is forwarded to the management process on the same host as the application process. The coordinator for the client would retrieve this policy on initialization and ensure that instrumentation components (sensors, in this case) are prepared to support this policy. When measured data indicates that the required conditions for the policy have been violated, the coordinator notifies the local quality of service resource management system process of the situation so that it may resolve the problem.

Sensors

A sensor encapsulates an attribute characterizing some element of application behaviour, measured according to some application metric. Sensors get their input data from probes inserted at strategic points in the application code or by reading other sensors. A sensor probe is an interface between process code and 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.

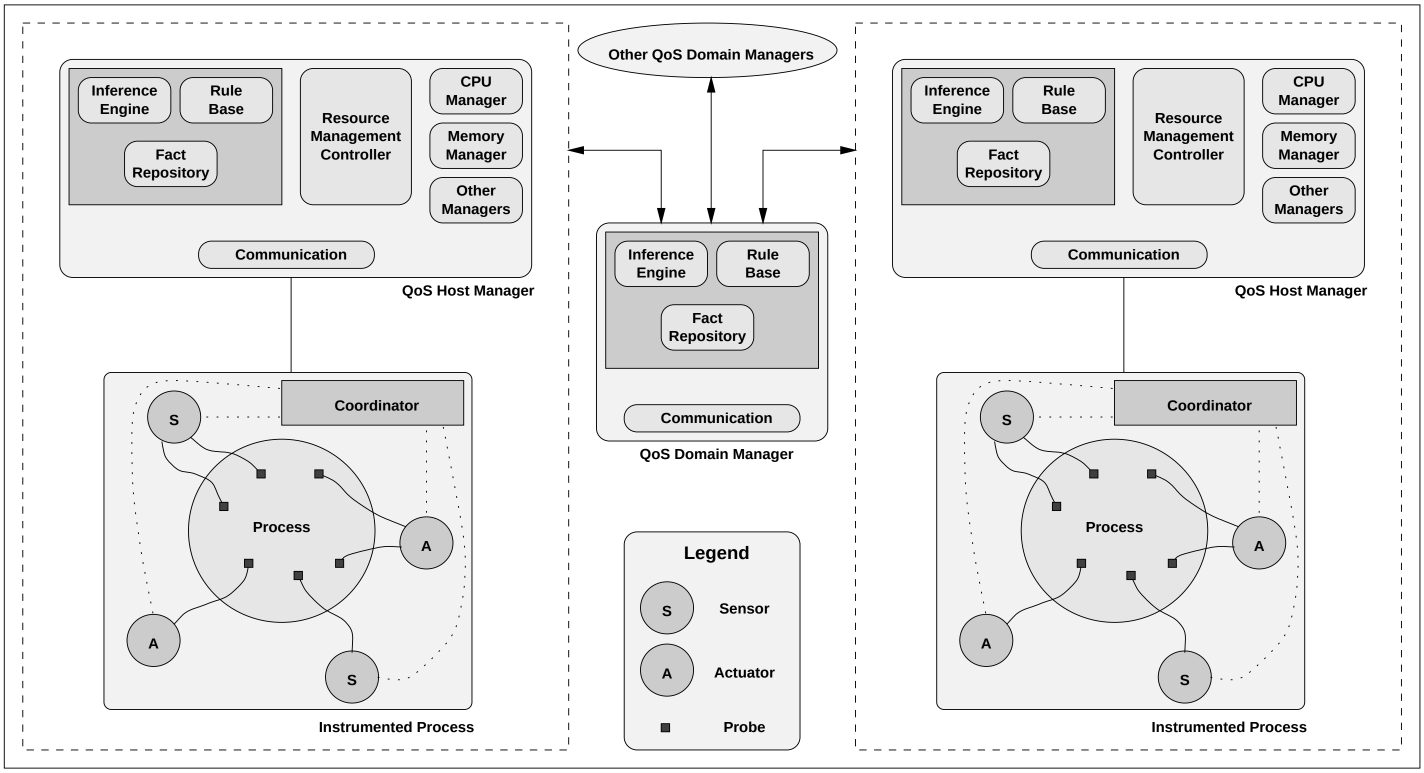

Figure 1: Usage Based Service Differentiation Architecture

Continuing the example from the previous section, a video frame rate sensor would be intialized by the coordinator. This sensor's thresholds would be set to report to the coordinator whenever the video frame rate falls below 22 frames per second, or exceeds 26 frames per second. As the sensor's probes provide input to the sensor, this data is filtered, analysed, and compared against thresholds. The coordinator is only notified when the calculated frame rate is not within threshold boundaries.

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.

Continuing our example from the previous sections, actuators may be put in place and initialized by the instrumentation coordinator to adjust video parameters in response to adaptation requests from the quality of service resource management system. For example, if insufficient resources exist to deliver quality above 22 frames per second, actuators may be used to reduce spatial resolution or choose a less computationally intense dithering method to free resources and bolster the delivered frame rate.

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.

Continuing the example from the previous sections, we would require probes for all of our sensors and actuators. The video frame rate sensor would require a probe in the code stream processing video frames to update the current frame rate in the sensor. The adaptation actuators adjusting video or processing characteristics would need probes to manipulate the appropriate data structures in the process to change the process's behaviour upon request.

3.2.2 Quality of Service Managers

In this section, we examine the internals of the quality of service managers, detailing the components used to support the usage based service differentiation process. The basic idea behind this architecture 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. Typically, this entails locating the sources of quality of service problems that are deemed to be not local to a single host, based on information collected from QoS Host Managers, and then deciding upon corrective actions to resolve the situation. A QoS Domain manager is only responsible for making such decisions, delegating the implementation of these decisions up to the appropriate QoS Host Managers. If problems span multiple such collections, additional coordination is necessary.

Having management processes on each host in this manner has several advantages over using a single centralized manager, despite incurring some resource overhead on each host. With a management process on every host, the quality of service management load is spread out, making the management system more scalable and more responsive. Further performance gains come from decreasing the physical distance and latency between the management system and managed process, and from being able to use more efficient local inter-process communication mechanisms, as opposed to remote communication mechanisms, which can be several orders of magnitude slower. Lastly, it is technically very difficult and incredibly insecure to allow remote applications to control local resource allocations. Consequently, this approach is the logical choice.

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. The fact base, rule base, and inference engine form the core of policy based management, while the resource management controller and resource managers provide dynamic end-to-end resource management, as discussed in Section 3.1. These components are discussed below in further detail.

Communication

The quality of service manager communication module is used to facilitate communication between the manager and other processes in the computing environment. Local interprocess 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 registered managed processes, including the process identifier, name, owner, start time, and so on.
- Data provided through quality of service violation reports, when quality of service no longer meets requirements, including the policy under examination, the process where the violation originated from, and the events leading to the violation, especially relevant process attributes collected by sensors.
- Behavioural user hints used to guide service differentiation decisions, including the current state of the application, time of last state change, idle time, and so on.
- Additional data queried from the operating system or other application processes as necessary.

Continuing our client-server video-on-demand example from previous sections, data on the application will be asserted into the fact base upon initialization, and on every violation reported. On initialization, the coordinator for each process registers the process, providing information such as the process identifier, name, owner, starting time, configuration, and so on, which is stored in the fact base. If the client's delivered frame rate falls below the expected 22 frames per second, or rises above 26 frames per second, the violation notification sent from the client process's coordinator to the local QoS Host Manager would result in the insertion of the following facts in the fact base:

- Frame rate violation notification.
- Time of the violation.
- Process identifier of the video client as the process in violation (can be matched with registration information).
- The target frame rate and actual delivered frame rate.
- Length of incoming video stream buffer to assist in determining the source of the problem.

Rule Base

The rule base contains a set of rules used 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, such as application design, network or system architecture, operating system design, administrative needs, and so on. Policies required by a QoS Host or Domain Manager are mapped to one or more rules loaded into the rule base.

For example, we continue the video-on-demand scenario from the previous sections. Rules pertaining to this application in the rule base could include the following:

- If the frame rate is too high, then the process should be given fewer resources, as determined by the resource management controller.
- If the frame rate is too low, and the client incoming buffer size is above some threshold (implying that the process is not able to handle incoming frames quickly enough), then the process should be given additional resources, as determined by the resource management controller.
- If the frame rate is too low, and the client incoming buffer size is below some other threshold (implying that the process is more than able to handle incoming frames quickly enough), then a notification should be sent to the QoS Domain Manager, as the problem likely lies in the network or the server system delaying frames to the client. Additional rules for the QoS Domain Manager would help locate the problem and formulate corrective actions.

To support usage based service differentiation, a variety of rules specific to this task are also 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 execute. The manipulations required are forwarded to the resource management controller, if resource adjustments are necessary, or to application processes, if other adaptations are deemed necessary.

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.

Resource managers can be generic or application specific. Generic resource managers can manage computing resources for any application, including CPU, network, disk, memory, and other resources. Application specific resource managers are geared towards tuning resource management for particular applications.

3.3 Bringing it All Together

To give an impression of how all the different architectural components from the previous sections function together, we discuss in this section how these components interact to manage and maintain quality of service under a differentiated services discipline.

1. As hosts in the computing environment are started, QoS Host Managers are executed on each host as part of the booting process. A QoS Domain Manager is also started on its designated host at boot time. In

this way, the quality of service management system is initialized along with the computing environment.

2. As applications start, they register with the QoS Host Manager local to their host, and this registration is forwarded to the QoS Domain Manager as well. This registration information is asserted into the fact base, as discussed earlier.

3. As users interact with their applications, this behaviour is recorded by user hint sensors. If this information is deemed significant (change in focus, viewability, usability, and so on), the sensors inform the application coordinator, which in turn contacts the local QoS Host Manager. The manager asserts appropriate facts into its fact base to reflect the state of the application in question, as discussed earlier.

   If an application is distributed, its QoS Host Manager will notify the QoS Domain Manager of significant changes in the state of the application, which will in turn propagate this information to the QoS Host Managers in charge of the other components of the application. By doing so, we can ensure that all of the necessary QoS Host Manager fact bases across a distributed computing environment contain the necessary behavioural user hint facts for quality of service management decisions concerning the distributed application.

4. When quality of service fails to meet requirements for an application, a sensor detects this condition, notifies its coordinator, which in turn notifies its QoS Host Manager. This manager, in turn, records facts surrounding this violation into its fact base, as discussed earlier, and uses its inference engine to determine how to correct the problem. If the problem is deemed to be local, resources are adjusted through the resource controller and resource managers accordingly. If the problem is not thought to be local, the QoS Domain Manager is contacted to resolve the situation using its own inferencing; the QoS Domain Manager contacts the appropriate QoS Host Managers to execute actions to resolve the problem.

5. When the supply of a resource is exhausted in attempting to resolve a quality of service requirement violation, a fact is asserted in the QoS Host Manager's fact base indicating service differentiation is to be applied. The presence of this fact causes resource management to follow different rules that give preferential access to resources to selected applications, and scales back allocations to others. The choice of applications is made through the use of additional rules—if facts pertaining to an application match the conditions of a rule, the application can be given either preferential treatment, left alone, or given substandard treatment, as specified by the matched rule.

   Behavioural user hint facts asserted above are used in this selection process, and can be matched in selection rules as any other application facts can. For example, if an application is minimized by its user, this state change is detected by a sensor and is eventually recorded as a fact in the QoS Host Manager's fact base, using the procedure above. When service differentiation is applied, the presence of this fact will cause the application to fail to match rules giving preferential treatment, and match rules requiring substandard treatment instead. Its resources will then be scaled back by the resource management controller and resource managers accordingly.

6. As applications terminate, or if quality of service requirements for an application decrease, the supply of available computing resources will increase. If service differentiation is currently being applied, and the supply of computing resources increases above a policy-set threshold, then service differentiation will be suspended by removing the fact asserted above from the fact base. If the supply of resources is later exhausted again, service differentiation can be imposed again, as described above.

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. For example, components are available to measure quality metrics from video applications (frame rate, frame loss, jitter), electronic commerce applications (transaction rates and response times), database applications (query rates and response times), and a variety of others.

User hint sensors have been provided to detect and handle generic behavioural user hints in an X Windows environment [36] (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). Proxy user hint sensors are also provided to handle the propagation of user hints through all processes of a distributed application.

As an example, consider the following sample function code from a greatly simplified video-on-demand client. It has been instrumented with both a frame rate sensor to monitor the displayed frame rate of streamed video received from its server, and a user hint sensor to monitor changes in the the state of the application's window.

```
1  status_t receive_and_display_video() {
2
3    Display *display;
4    Window *window;
5    int frame_number = 0;
6    VideoFrame *frame;
7    X11_user_hint_sensor uh_sensor;
8    frame_rate_sensor fps_sensor;
9
10   /* Initialize X11 display and window. */
11
12   initialize_display(&display, &window);
13   uh_sensor->probe(display, window);
14
15   /* Loop through receiving video frames */
16   /* and displaying them until video done. */
17
18   while(video_not_done()) {
19     frame = receive_frame(frame_number++);
20     display_frame(display, window, frame);
21     fps_sensor->probe();
22   }
23
24   return OK;
25 }
```

Line 1 declares our simple function, the purpose of which is to receive and display video frames from a video-on-demand server. For our purposes here, we assume that the server has already been contacted and that the desired video has been selected and is queued for transmission. Lines 3 and 4 declare display and window variables used by X11 to indicate where the video should be displayed. Lines 5 and 6 declare variables to indicate the sequence number of the video frame currently being processed, and a pointer to a video frame handle used to refer to a video frame. Lines 7 and 8 declare our user hint and video frame rate sensors respectively. Line 12 calls a function to initialize X11 for displaying the video, and retrieves the display and window handles to be used. Line 13 calls a probe to our user hint sensor to register the video window for tracking of X11 user hints, as described above. Whenever a change in the state of the video window occurs, this sensor will now be able to detect and report these changes. Line 18 creates a loop to process all video frames until the video is finished. Line 19 invokes a function to receive the appropriate video frame, and increments the frame counter to retrieve the next frame. Line 20 invokes another function to display the received video frame to the X11 window. Line 21 concludes the loop by calling a probe to our frame rate sensor. Internal sensor code records probe calls, applies filtering, and calculates the current frame rate of displayed video, measured in frames per second. If this frame rate violates policy-set thresholds, the instrumentation coordinator for the client will be notified, as discussed in Section 3, to resolve the situation. Line 24 concludes the function by returning a successful status.

The instrumentation library implemented as part of this work can be further specialized to develop new classes of instrumentation components to meet the unique needs of specific applications. This is done through providing a variety of abstract classes suitable for specialization. Further implementation details of our instrumentation and additional code examples can be found in [23].

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 resource management controller supporting a variety of algorithms that work with a collection of resource managers that each manage a single system resource. To date, we have resource managers capable of adjusting CPU allocations (through manipulating time-sharing priorities, or by allocating units of real-time CPU cycles) and adjusting memory allocations (through adjusting the number of resident pages each process has in physical memory). Network resource adjustment is currently being added to our prototype [5]. 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 [16]. 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.

The CLIPS fact base is represented as a collection of tuples of information, with one fact per tuple. Virtually any tuple of information can be a valid CLIPS fact, though it is generally wise to place meaningful information in each tuple to ensure it is humanly readable. Every fact has the following general form:

```
( field_1 field_2 ... field_n )
```

In this case, each field is filled in by some appropriate CLIPS symbol, including text, numbers, and so on. For additional information on the CLIPS facts, refer to [16].

Rules in the rule base are expressed as a left-hand-side and a right-hand-side. The left-hand-side lists the fact patterns that must be matched for the rule to be activated. Fact patterns can either be complete facts, specifying the contents of each field, or partial facts, where some fields are left as variables to be filled in by the matching process. The right-hand-side lists the actions to be carried out when the rule is activated. The general form for a rule is as follows:

```
(defrule rule_name "optional_comment"
   (pattern_1)
   (pattern_2)
       .
       .
       .
   (pattern_n)
=>
   (action_1)
   (action_2)
       .
       .
       .
   (action_m)
)
```

The special text => is used to split the fact patterns in the left-hand-side from actions in the right-hand-side. All rules must provide a header beginning with the `defrule` token, giving the name of the rule, as well as an optional comment describing the rule. The entire rule must be enclosed within matching parentheses. There are other rule formatting guidelines restrictions; refer to [16] for further details.

Rules written in this CLIPS language are compiled into a binary format for quicker execution. Actions in rules can either be written in CLIPS, or functions written in C or C++. While greater usage of CLIPS provides better runtime flexibility and dynamic characteristics, C or C++ code executes with greater efficiency and performance. Consequently, a balance must be struck accordingly.

Further implementation details of the QoS Host and QoS Domain Managers can again be found in [23].

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 runtime 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.

4.3.1 Example: Triggering Service Differentiation

An example of a resource management policy triggering service differentiation is given below. Essentially, this is a diagnostic policy that is triggered in response to a low frame rate from a video-based application. It is diagnostic in the sense that it determines which resources should be allocated to the application and by how much. It also checks to ensure that such allocations are possible; if not, service differentiation is put into effect.

```
(defrule low_frame_rate
   (registered process_id ?pid)
   (fps low ?current_fps ?target_fps)
=>
```

```
5      (bind ?amount (quality_distance
6        ?current_fps ?target_fps 10))
7
8      (if (> (prob ?pid cpu ?amount) 0.5)
9        then
10       (if (< (poss ?pid cpu ?amount) 0.1)
11         then
12         (assert (service_diff cpu))
13         else
14         (adjust_resource ?pid cpu ?amount)
15       )
16     )
17
18     (if (> (prob ?pid mem ?amount) 0.5)
19       then
20       (if (< (poss ?pid mem ?amount) 0.1)
21         then
22         (assert (service_diff mem))
23         else
24         (adjust_resource ?pid mem ?amount)
25       )
26     )
27
28     (set_reset_delay 1)
29  )
```

For this rule to activate, the fact patterns on lines 2 and 3 must be matched. In other words, we require the process identifier of the application process whose quality of service policy is being violated, and we require that the violation resulted from a frame rate delivered below its target value. (Both pieces of information are asserted into the fact base when the quality of service violation is reported by the application process's coordinator.) Lines 5 and 6 then determine the amount of resource allocation required using a function `quality_distance` that we have added to CLIPS, based on the current frame rate, the target frame rate, and a scaling factor selected as 10 in the rule above. Lines 8–16 analyse the need for and availability of additional CPU resources. Line 8 uses another function added to CLIPS, `prob`, to determine the probability that allocating the given amount of CPU resources will improve the quality of the application in question. (This function interacts with the resource management controller and resource managers to make this determination based on application resource consumption and other system measurements.) If it is greater than 50%, line 10 then checks to see how possible it is to make such an allocation, given the remaining supply of the resource. This check is done using another function added to CLIPS, `poss`. (This function also interacts with the resource management controller and resource managers to make this determination.) If chances are less than 10determines that CPU resources have been exhausted, and identifies that service differentiation should be applied with respect to this resource by adding a new fact to the fact base; otherwise, the resource adjustment is made using another function added to CLIPS, `adjust_resource`. (This function also interacts with the resource management controller and resource managers to make this determination.) Lines 18–26 perform analagous steps for memory resources instead. Finally, line 28 indicates that the QoS Host Manager should not accept further violations from this process for 1 second (to allow for the adjustment to take effect before further complaints are heard).

4.3.2 Example: Simple Service Differentiation Policy

An example of a simple service differentiation policy is given below. Essentially, this rule states that if a video-on-demand application reports a violation of quality of service requirements, and encounters a shortage of resources in doing so, then all applications started on the same host afterwards should be penalized. In this case, such applications are penalized by losing their allocations of the resource in question, and essentially become best effort applications in that respect. This provides preferential service to applications starting earlier, at the expense of applications started afterwards.

```
1   (defrule service_differentiation
2      (service_diff ?resource)
3      (registered process_id ?fpid)
4      (reginfo ?fpid ?time vod ?user ?host)
5      (reginfo ?rpid ?time2 ?app2 ?user2 ?host)
6      (service_diff_flag ?rpid ~reduced_service)
7   =>
8      (if (> ?time2 ?time)
9        then
10       (set_diff_flag ?rpid reduced_service
11         ?host)
12       (remove_resources ?rpid ?resource)
13       (action_directive ?host ?rpid
14        "coordinator->disable_all()" CLIPS_FACT
             "(disable sensors)")
15     )
16  )
```

Line 2 of this rule ensures that this rule is only activated when some resource has been flagged for service differentiation due to a resource shortage. Line 3 obtains the process identifier of the process reporting the quality of service violation. Line 4 matches this to registration information so we can obtain the name of the application, its start time, and the host it is executing on. In this way, we can ensure that this rule is only activated when the process reporting the violation of quality of service requirements is part of a video-on-demand application. Line 5 is then used to match all other processes executing on the same host. Line 6 ensures that the other application processes are not already marked for reduced service. Line 8 is used to filter out all processes that started before or at the same time as the process reporting the violation, so we are left with only those that started afterwards. Lines 10 and 11 flag the application to receive reduced service as such, using the `set_diff_flag` function added to CLIPS. Line 12 then removes its resource allocation, using another function added to CLIPS, `remove_resources`. Lastly, lines 13–15 instruct the coordinator for the application process receiving reduced service to disable its sensors to prevent the process from complaining that its resources have been taken away, and that it is in violation of its quality of service requirements. This is done though another function added to CLIPS, `action_directive`.

4.3.3 Example: Usage Based Service Differentiation Policy

An example of a usage based service differentiation policy is given below. Essentially, this rule states that if service differentiation has been enacted, all unviewable applications are to be flagged for reduced services, by removing resource allocations given to the application and disabling instrumentation in the application. This essentially relegates unviewable applications to best effort service.

```
1  (defrule usage_based_service_differentiation
2     (service_diff ?resource)
3     (userhint ?pid unviewable)
4     (reginfo ?pid ?time ?app ?user ?host)
5     (service_diff_flag ?pid ~reduced_service)
6  =>
7     (set_diff_flag ?pid reduced_service ?host)
8     (remove_resources ?pid ?resource)
9     (action_directive ?host ?pid
10     "coordinator->disable_all()" CLIPS_FACT
           "(disable sensors)")
11    )
12 )
```

Line 1 of this rule provides a declaration and name for this rule, while lines 2–5 provide conditions to be matched for the rule to execute, and lines 7–11 list the actions to be executed when matching has occured. Line 2 ensures that this rule is only activated when some resource has been flagged for service differentiation due to a resource shortage. Line 3 obtains the process identifier of the process and ensures that behavioural user hints indicate the application is unviewable. Line 4 matches this identifier to registration information so we can obtain the name of the host it is executing on for future use. Line 5 ensures that the application is not already marked for reduced service. Line 7 flags the process for reduced service, which also inserts the fact checked on line 5 into the fact base. Line 8 cancels resource allocations made to the process, forcing the application to best effort status. Lastly, lines 9–11 contact the coordinator for the process and direct it to disable its sensors to prevent the application from submitting future reports that it is missing its quality requirements.

Additional implementation details and examples of the service differentiation policies used in this work can again be found in [23].

5 Experimental Results and Experience

In the previous section, we discussed the implementation of our prototype quality of service management system for usage based service differentiation. In this section, we present results from experiments conducted using this prototype system.

For experimentation in this section, 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 these experiments: `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.

5.1 No Management Experiments

Before experimenting with service differentiation policies, we first carried out initial experiments to gauge the performance of the application without management in place. This includes executing a single instance of the application, as well as multiple instances.

Figure 2 presents a representative run from a single instance of the video-on-demand application, with the server executing on `farquarson`, and the client receiving and displaying video executing on `strawberry`. The application had no difficulty in delivering acceptable quality, after only

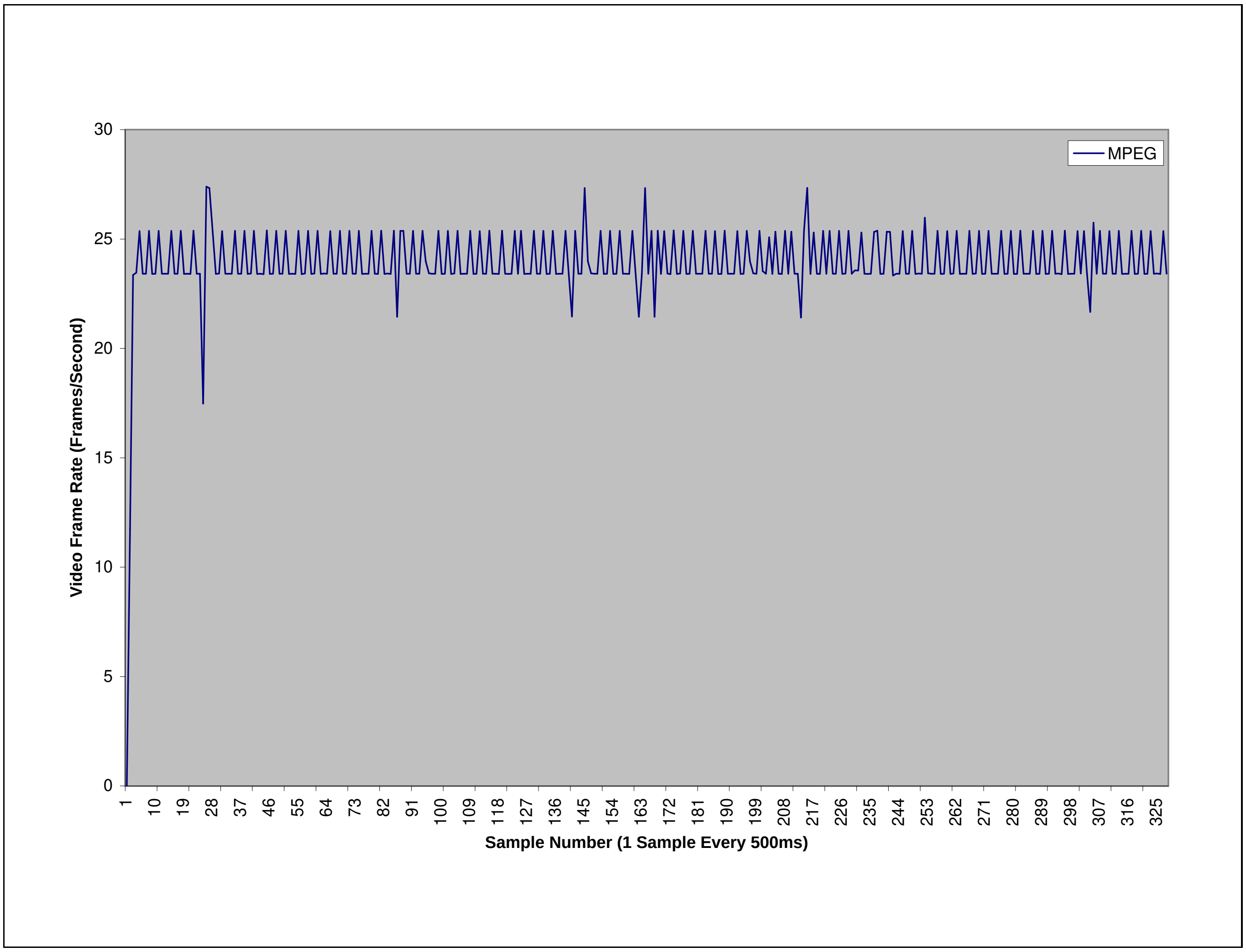

Figure 2: Single Video-on-Demand Application Experiments

1.23% of its run and missing requirements only 3.68% of the time after that. A mean frame rate of 23.91 frames per second was delivered on average, with a standard deviation of 1.31. It was noted during experimentation, however, that a single instance of the application consumed roughly 83% of CPU cycles on the client end; consequently, multiple instances of the application could present difficulties in meeting quality objectives.

Figure 3 presents a representative run from multiple instances of the video-on-demand application. In this case, servers were executed on `farquarson` and `vanilla`, while both clients were executed on `strawberry`. (A second server host was used to avoid disk contention problems observed in experiments discussed in [23].) As was expected, quality of service was quite poor and highly variable. The mean frame delivered was only 9.68 frames per second, with a standard deviation of 6.51. Quality of service requirements were violated 91.9% of the time. In short, the computing environment could simply not support multiple instances of the video-on-demand application with the given video workload.

5.2 Uniform Service Experiments

Before examining the effects of service differentiation under this scenario, we first demonstrate how management using a uniform services discipline fails to achieve acceptable quality for either instance of the video-on-demand application.

Figure 4 shows the execution of both instances of the

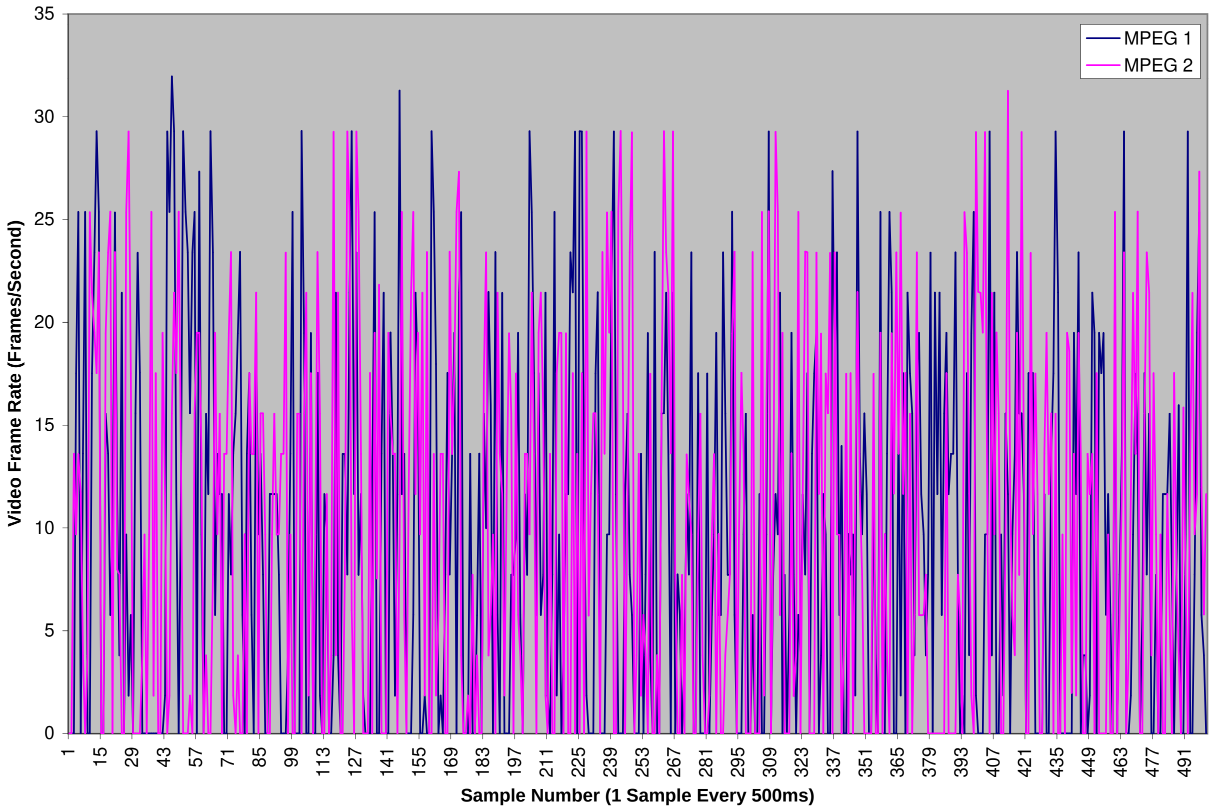

Figure 3: Multiple Video-on-Demand Application Experiments

video-on-demand application, configured the same as in the previous experiment, except with our quality of service management system in place. Our management system was configured to use a uniform services discipline, in which access to resources was on a first-come, first serve basis, without discrimination. While quality of service delivered marks a substantial improvement over the previous case in which no management was available, both application instances failed to meet their quality of service requirements 100% of the time. Instead, both applications acquired what resources they could, and did their best after the supply of resources was exhausted.

5.3 Differentiated Service Experiments

For this experiment, a service differentiation policy was in place dictating that the first instance of the video-on-demand application to start execution should be favoured and receive resources to meet quality of service constraints. Other applications suffering as a result of this favouring were still required to deliver quality at least at 50% of their original quality of service requirements. For these experiments, this meant that the second instance of the video-on-demand application must deliver quality of at least 12 frames per second, with the same 2 frame per second deviation allowed on the lower end (to 10 frames per second), and an the same upper bound as before (26 frames per second).

Figure 5 shows a representative run in this scenario with

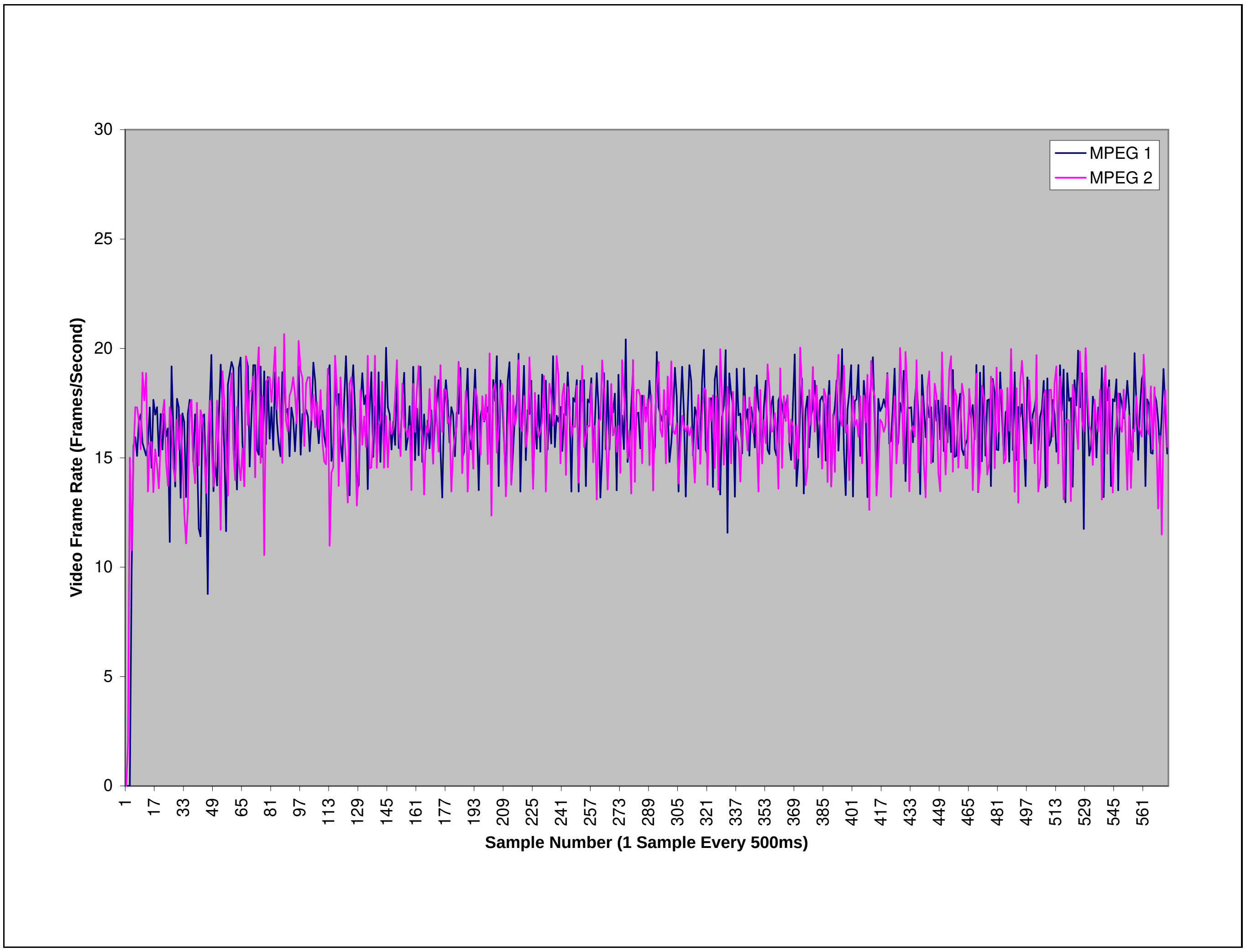

Figure 4: Video-on-Demand with a Uniform Services Discipline

the service differentiation policy in place. The quality of service management system enacted service differentiation as soon as the supply of resources on the host was exhausted, yet demand for the resources was still high. The management system scaled back resources allocated to the second instance of the application and reallocated them to the first. Consequently, the first instance was able to converge and deliver stable quality meeting quality of service requirements, while the second was still able to to meet its 50% quality requirement.

It is interesting to note that during these experiments, however, that behavioural user hints from the applications were ignored. At many points during experimentation, the first application was covered, or minimized, yet this did not change resource consumption or quality delivered by either application. This, naturally, is unfair to the second instance of the application which suffered needlessly when the video displayed by the first instance was unviewable.

5.4 Usage Based Differentiated Service Experiments

In this experiment, we used the same service differentiation policy as in the previous section. 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 dis-

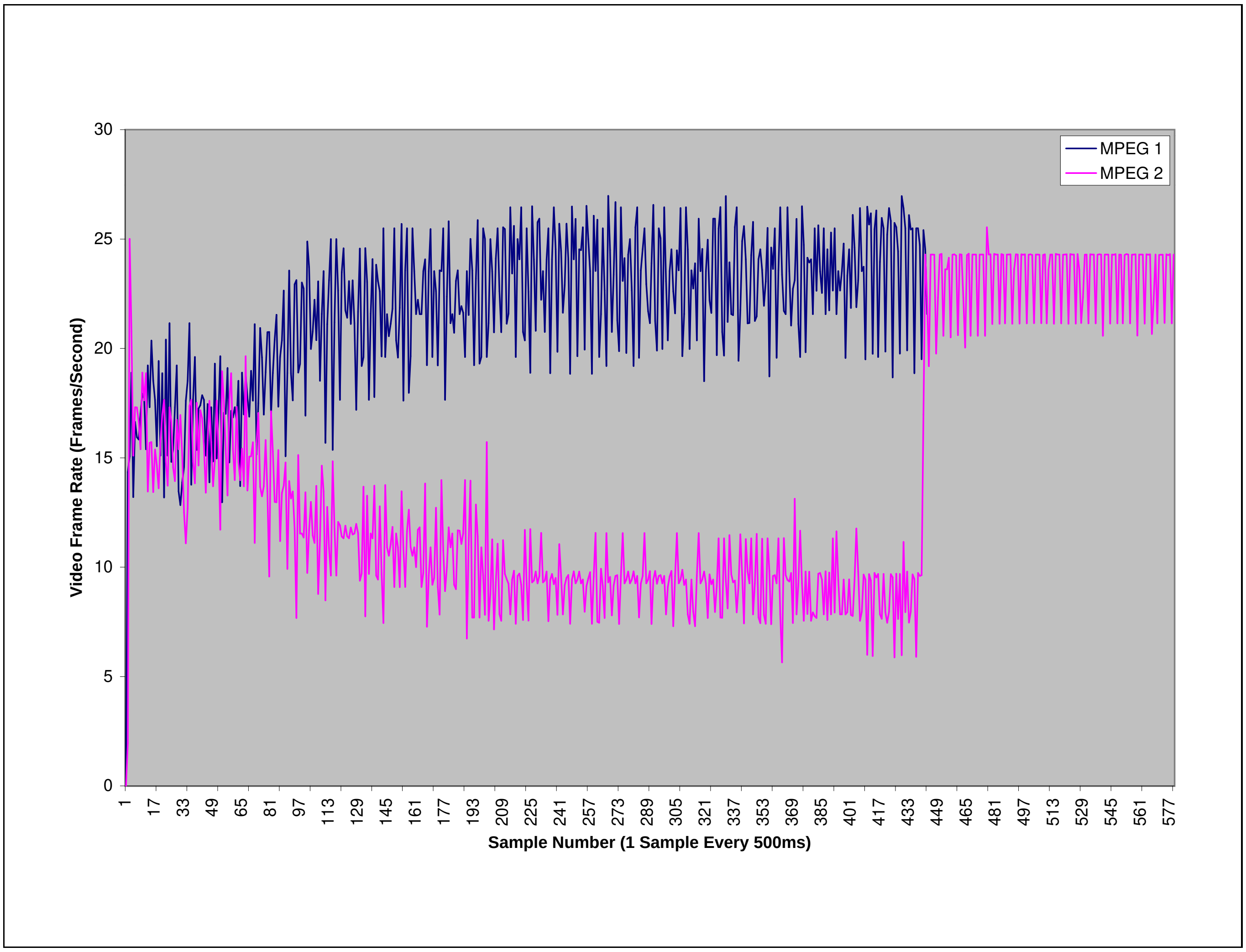

Figure 5: Video-on-Demand with a Differentiated Services Discipline

criminators used in service differentiation decisions.

Figure 6 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

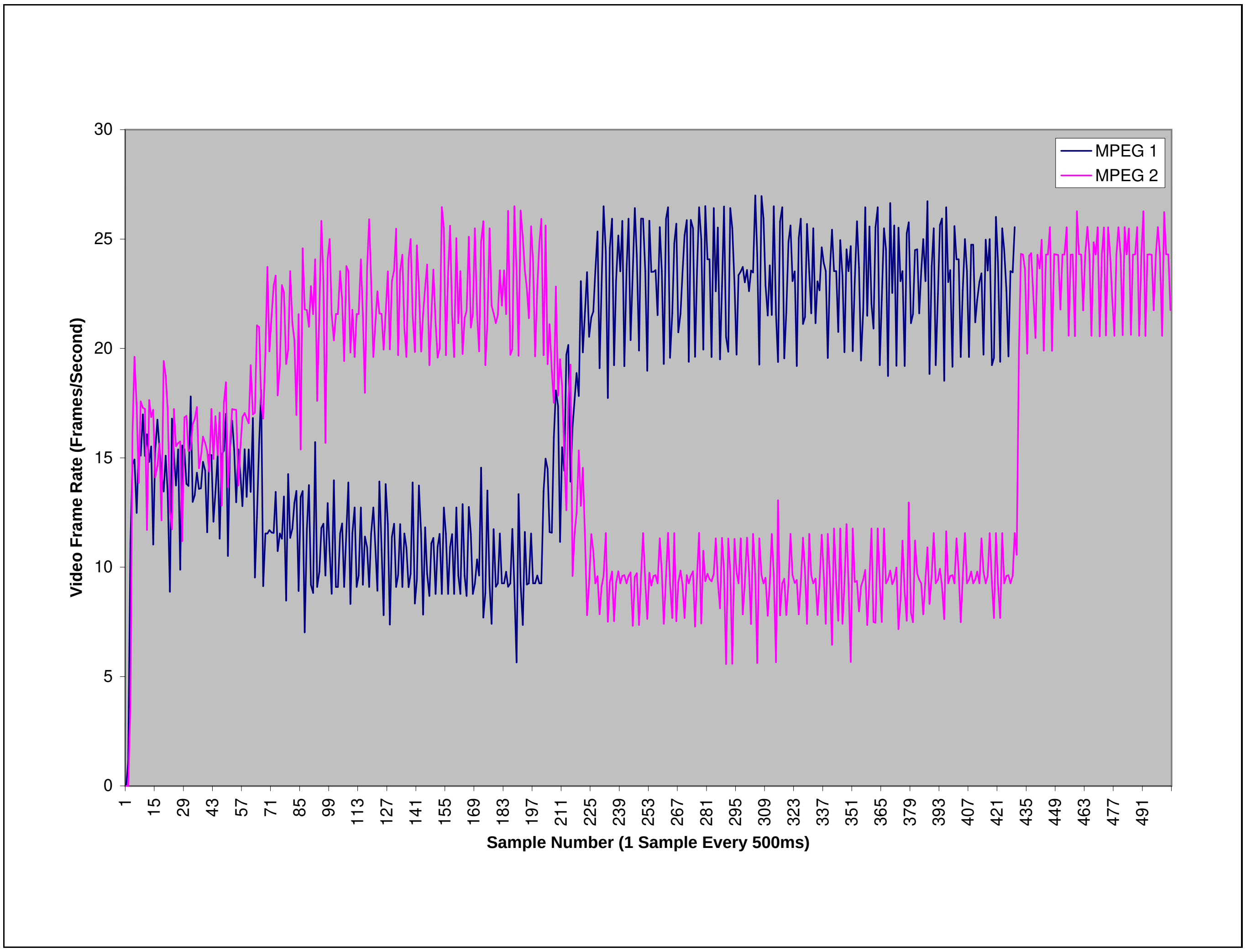

Figure 6: Video-on-Demand with Usage Based Differentiated Services

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. This includes generic information applicable to a variety of applications, as well as application-specific information geared towards either certain classes of applications or individual applications.

- 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 adding management 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 effectively and 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 SIGMETRICS 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] L. Bannon, A. Cypher, S. Greenspan, and M. Monty. Evaluation and Analysis of Users' Activity Organization. *Proceedings of the 1983 ACM Conference on Human Factors in Computing Systems*, Boston, Massachusetts, December 1983.

[5] 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.

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

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

[8] 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.

[9] B. Davison and H. Hirsh. Predicting Sequences of User Actions. *Proceedings of the AAAI/ICML 1998 Workshop on Predicting the Future: AI Approaches to Time-Series Analysis*, New Brunswick, New Jersey, July 1998.

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

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

[12] F. Le Faucheur, L. Wu, B. Davie, S. Davari, P. Vaananen, R. Krishnan, P. Cheval, and J. Heinanen. Multi-Protocol Label Switching (MPLS) Support of Differentiated Services. *IETF RFC 3270*, May 2002.

[13] K. Gaylin. How are Windows Used? Some Notes on Creating an Empirically-Based Windowing Benchmark Task. *Proceedings of the 1986 ACM Conference on Human Factors in Computing Systems*, Boston, Massachusetts, April 1986.

[14] M. Gervasio, W. Iba, and P. Langley. Learning User Evaluation Functions for Adaptive Scheduling Assistance. *Proceedings of the Sixteenth International Conference on Machine Learning*, Bled, Slovenia, June 1999.

[15] M. Gervasio, W. Iba, and P. Langley. Learning to Predict User Operations for Adaptive Scheduling. *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, Madison, Wisconsin, July 1998.

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

[17] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured Forwarding PHB Group. *IETF RFC 2597*, June 1999.

[18] E. Horvitz. Thinking Ahead: Continual Computation Policies for Allocating Idle and Real-Time Resources to Solve Future Challenges. *Proceedings of the Sixteenth Joint Conference on Artificial Intelligence*, Stockholm, Sweden, July 1999.

[19] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse. The Lumiere Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users. *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, Madison, Wisconsin, July 1998.

[20] E. Horvitz, A. Jacobs, and D. Hovel. Attention-Sensitive Alerting. *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, Stockholm, Sweden, July 1999.

[21] V. Jacobson, K. Nichols, and K. Poduri. An Expedited Forwarding PHB. *IETF RFC 2598*, June 1999.

[22] E. Kandogan and B. Shneiderman. Elastic Windows: Evaluation of Multi-Window Operations. *Proceedings of the 1997 ACM Conference on Human Factors in Computing Systems*, Atlanta, Georgia, March 1997.

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

[24] M. Katchabaw, H. Lutfiyya, and M. 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.

[25] M. Katchabaw, H. Lutfiyya, and M. Bauer. A Survey of Windowing Environment Usage Patterns. *Technical Report #540, Department of Computer Science, The University of Western Ontario*, May 1999.

[26] B. Korvemaker and R. Greiner. Predicting Unix Command Lines: Adjusting to User Patterns. *Proceedings of the 17th National Conference on Artificial Intelligence*, Austin, Texas, July 2000.

[27] H. Lutfiyya, G. Molenkamp, M. Katchabaw, and M. Bauer. Issues in Managing Soft QoS Requirements in Distributed Systems Using a Policy-Based Framework. *Proceedings of the International Workshop on Policies*, Bristol, England, January 2001.

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

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

[30] G. Molenkamp, M. Katchabaw, H. Lutfiyya, and M. 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.

[31] G. Molenkamp, M. Katchabaw, H. Lutfiyya, and M. Bauer. Managing Soft QoS Requirements in Distributed Systems. *Proceedings of the Multimedia Systems Workshop (ICPP)*, Toronto, Canada, August 2000.

[32] H. Motoda, T. Washio, T. Kayama, and K. Yoshida. Extracting Behavioral Patterns from Relational History Data. *Proceedings of the Sixth International Conference on User Modeling*, Chia Laguna, Sardinia, June 1997.

[33] 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.

[34] 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.

[35] 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.

[36] L. Reiss and J. Radin. *X Window Inside and Out.* McGraw Hill Incorporated, 1992.

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

[38] Sun Microsystems Inc. *The Complete Guide to ShowMe 2.0.1.* Mountain View, California, May 1994.

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

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

[41] A. Vogel, B. Kerhevré, G. V. Bochmann, and J. Gecsei. Distributed Multimedia Applications and Quality of Service: A Survey. *Proceedings of the 1994 Centre for Advanced Studies Confererence*, Toronto, Canada, November 1994.

[42] J. Walpole, R. Koster, S. Cen, C. Cowan, D. Maier, D. McNamee, C. Pu, D. Steere, and L. Yu. A Player for Adaptive MPEG Video Streaming Over The Internet. *Proceedings of the 26th Applied Imagery Pattern Recognition Workshop*, Washington DC, October 1997.