Adding Realtime Applets and Quality of Service Support to the World Wide Web*

K. Lakshman, Madan Manoharan†
University of Kentucky
lakshman,madan@dcs.uky.edu

Raj Yavatkar‡
Intel Corporation
yavatkar@ibeam.intel.com

Abstract

Predictable Quality of Service (QoS) in terms of controlled response time and guaranteed bandwidth is increasingly important in World Wide Web. We propose a variety of enhancements and extensions to the existing Web infrastructure so that Web applications and applets can provide predictable quality of service. The proposed extensions include addition of the notion of QoS to the HTML/HTTP/Java syntax, QoS negotiation and adaptation mechanisms in Java classes and threads, QoS reservation support at the socket and transport protocol layers, and ATM signaling extensions for dynamic QoS re-negotiation over persistent HTTP connections. Our ongoing work includes implementation of these extensions and an evaluation over a testbed consisting of a network of Unix workstations and ATM switches.

1 Introduction

The *Web* (World-Wide Web) is being increasingly used as a tool for delivering a wide variety of information, for commercial transactions, and for enhancing productivity (group-based collaboration, telephony, distance learning and training, and teleconferencing) within both corporate *intranets* and across the Internet. *Visiting a Web page* now typically involves accessing and displaying rich content consisting of text, graphics, animation, and audio/video streams. Predictable Quality of Service (QoS) in terms of both controlled response time and guaranteed bandwidth is increasingly important for ensuring high-quality delivery of rich media content. Furthermore, addition of Java-based applets [7] to the Web content imposes new demands on management of OS resources. For instance, a real-time applet designed to create an animated, interactive display involving audio, video, and images needs OS support for negotiating a predictable QoS and access to target host resources such as CPU, memory, disk, network, and bus bandwidth. Because the resource availability and host capabilities in a target environment are not known in advance, an applet must also be designed to adapt its behavior to achieve best possible QoS with resources available on a target machine.

We are currently investigating alternatives for adding QoS support to the Web infrastructure. While a considerable amount of work has been done in the area of OS support for multimedia [3, 4, 5, 6], very little work has been reported [1] in the area of creating the necessary infrastructure for providing real-time support to the Web-based applications.

Our current work grew out of the AQUA [11] (Adaptive QUality of service Architecture) project at the University of Kentucky. AQUA provides an OS framework for managing host resources (such as CPU, memory, and bus bandwidth) so that an OS can provide predictable service to applications. In particular, AQUA does not require applications to know their resource requirements in advance and allows applications (and OS) to adapt to changes in demand for host resources.

We will address the following issues in the rest of this paper:

Need for mechanisms to specify the QoS Requirements: To allow the specification of QoS requirements, WWW programming tools and interfaces such as HTML, HTTP, the Java language, and BSD Unix Socket library API must be extended. For instance, to associate QoS with applets, Java must support QoS as first class objects that can be passed to the underlying protocol stack.

Provision for alternate forms of content to accommodate heterogeneous networks: Despite the availability of new high-speed technologies, the Internet will continue to be a collection of heterogeneous networks with widely varying capabilities. Thus, the Web infrastructure must accommodate a home

*This work was supported in part by NSF grants STI-9108764 and CDA-950264.
†Supported by the Center for Computational Sciences, University of Kentucky.
‡On leave of absence from the University of Kentucky.

user connected to Internet across a 28.8 Kbps modem as well as a business user accessing a Web page across fast Ethernet and ATM links. Therefore, it must be possible for an author to offer alternate forms of content for the same link (or component) of a hypermedia document. For example, a component might be served in three alternate forms, namely, *text-only*, *text and graphics*, and *text and graphics and animation*. A Web client would then automatically choose (possibly without any user intervention) the right form of content based on the QoS capabilities of its connection to the server. Similarly, for audio and video clips, choice of content in alternate encodings (MPEG2 high-quality video vs. a low-bitrate H.263 stream, or a low-quality audio channel vs. Hi-Fi, stereo audio) should be offered with varying QoS requirements. These encodings may be generated automatically from the original content (available in one of the encodings), but HTML syntax and semantics must be extended to offer web clients alternate forms of content for the same URL and to associate different QoS specifications with each alternative. Moreover, in the case of video and animation, it should be possible to structure each alternate form as a series of layers using media-dependent hierarchical encoding.

OS Support for real-time, adaptive threads in Java: Java allows the creation of applets that are designed independent of a specific OS or hardware architecture. However, applets that perform real-time animations or transfer and playback of audio/video streams need QoS support from a host operating system. Moreover, since the capabilities and resource availability of target host OS and architecture are not known in advance, such applets must be able to dynamically negotiate necessary QoS with the host OS and adapt to operate within available resources. Thus, Java must be extended to include a notion of real-time threads and to include well-defined interfaces for QoS adaptation and negotiation with the host OS.

QoS support at the transport protocol layer: Both TCP and UDP are likely to be used as the transport protocols for transfer of data associated with Web documents. However, currently, these protocols rely on best-effort datagram delivery and do not allow applications to explicitly specify resource requirements and take advantage of QoS guarantees supported by underlying link and network layer technologies (e.g., ATM or RSVP). Appropriate QoS API must be added to the transport layer interfaces [16] and/or transport protocol implementations should be extended to transparently establish QoS reservations on behalf of the applications.

Dynamic QoS re-negotiation over persistent HTTP connections: Recently, it has been proposed that HTTP use persistent connections [13] to carry multiple requests to the same server for efficient usage of network resources and to reduce latency. However, the QoS requirements of each such request is likely to vary widely depending on the type of content requested and support should be provided to allow dynamic renegotiation of QoS reservation over an ongoing connection.

We have built a prototype implementation of *QoS-enabled* Web infrastructure consisting of API extensions to Java, HTML, BSD socket library, addition of real-time threads and interfaces for QoS negotiation/adaptation to Java, transport-level support for mapping TCP/UDP data flows to ATM-based (and possibly RSVP-based) QoS guarantees, and dynamic QoS renegotiation for persistent HTTP connections.

Rest of this paper describes our proposals in detail. Section 2 presents extensions to allow QoS specifications, Section 3 discusses methods to provide alternate content forms, and Section 4 describes extensions to Java to write real-time applets that can dynamically negotiate necessary QoS and adapt to available QoS. Sections 5 and 6 respectively summarize the proposed extensions to TCP/IP stack for QoS mapping and dynamic QoS renegotiations over a persistent HTTP connection.

2 Mechanisms to Specify QoS Requirements

To allow Web-based applications and applets to specify their QoS requirements, we suggest following extensions and enhancements to HTTP, HTML, Java, and the Unix socket interface.

QoS Specifications in HTML: Currently, QoS specifications in a document are limited to attributes such as *width* and *height* of an image. However, such hints are rudimentary at best and cannot be effectively

```
QoS Class
public class QoS {

        QoSRange peak_bandwidth;
        QoSRange mean_bandwidth;
        QoSRange mean_burst;

    public QoS(int peak, int mean, int burst) {
        this.peak_bandwidth = new QoSRange(peak);
        this.mean_bandwidth = new QoSRange(mean);
        this.mean_burst = new QoSRange(burst);
    }

    public QoS(int peak) {
        this.peak_bandwidth = new QoSRange(peak);
        this.mean_bandwidth = new QoSRange(peak);
        this.mean_burst = new QoSRange(default);
    }
}
```

```
URL with QoS
public class QoSURL extends URL {
    String protocol;
    String host;
    String file;
    int port;

    QoS qos;

    ......
    constructor and handler methods
    .......
}
```

```
Socket with QoS
class QoSSocketImpl extends PlainSocketImpl {
    public native void setQoS(QoS qos);

    static {
        System.loadLibrary("QoS");
    }
}
```

```
An applet that retrieves data with bounded time
public class QoSGetData extends java.applet.Applet
    implements Runnable {
    QoSURL theURL;
    Thread runner;

    public void init() {
        String url = "http://banjo.dcs.uky.edu:8001/data.file";
        this.theURL = new QoSURL(url);
        this.theURL.setTime(1000); /* 1 sec */
    }

.......

    public void run() {
        InputStream conn = null;
        DataInputStream data = null;
        int length = this.theURL.openConnection().getContentLength();
        /* set peak bandwidth Kbps in QoS object */
        qos = new QoS(length/(url.getTime() * 1024));
        this.theURL.setQoS(qos);
        conn = this.theURL.openStreamTime();
        data = new DataInputStream(new BufferedInputStream(conn));
        /* perform data operations */
    }
}
```

Figure 1: Sample code in Java-like syntax shows extended Java classes to accommodate QoS requests. QOSGetData is a simple applet designed to receive and process data with delay no more than one second. URL and several other classes used to provide QoS extensions are final classes in standard Java and do not allow subclassing, the actual implementation copies the methods of such classes.

used for deriving QoS requirements (bit rate, latency bounds, playback rate, etc) for transport of more complex objects such as applets, embedded objects, animations, and audio/video clips. We propose addition of a standard QoS attribute for all objects. Our current implementation uses a QoS specification for such attributes based on the Fore Systems ATM SPANS signaling API.

QoS extensions to HTTP: QoS-based extensions to HTPP are necessary to allow specification of QoS requirements to the underlying transport as part of a *get* request. Extensions to HTTP proposed in [1] allow content-based searches and interactive control over playback operations.

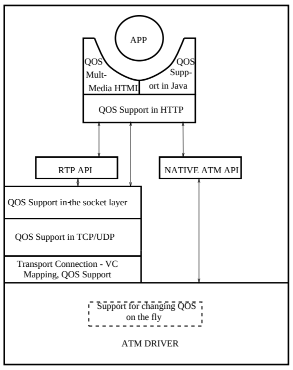
We have modified the HTTP protocol so that a Web client can optionally specify that a particular HTML document be fetched with a QoS reservation. We have implemented our modifications using the Web server available from the Web (W3C) Consortium. For example, a `GetWithQos` command allows an application to specify the QoS requirements and the transport level source address as shown in following examples:

```
GETWITHQOS movie.mpg
TCP 4444 MEAN_BW 8000 Kbps PEAK_BW 9000 Kbps QHTTP/1.0

GETWITHQOS movie.mpg
UDP 4444 MEAN_BW 8000 Kbps PEAK_BW 9000 Kbps QHTTP/1.0
```

As described in Section 5, we have extended an Unix TCP/IP implementation to set up QoS reservations in response to QoS specification from HTTP clients.

Extensions to Java and the BSD Unix Socket Library: Given the support for QOS specification in HTML and in the HTTP protocol, the next step is to make provision for Web clients and applications



<insert name="newsclip">
<altdata src="clipl1.mpg" id="mpg" rchlevel=1 type="application/mpg" qos_peak=1.5 Mbps>
<overlay src="clipl2.mpg" pid = "mpg" type="application/mpg" qos_peak=0.7 Mbps>
<overlay src="clipl3.mpg" pid = "mpg" type="application/mpg" qos_peak=0.5 Mbps>
<overlay src="clipl4.mpg" pid = "mpg" type="application/mpg" qos_peak=0.3 Mbps>
<altdata src="clip.h263" id = "h.263" rchlevel=2 type="application/h263" qos=clip.h263.file>
<altdata src=photobig.gif alt="The photo" rchlevel=4 type="application/xv qos_fromlength>
<altdata src=phototumb.gif alt="The photo" rchlevel=5 type="application/xv qos_fromlength>
<altdata src="The news clip shows the signing of the peace treaty" type="inlinetext" rchlevel=6>
</insert>

Figure 2: Proposed protocol architecture and API interfaces are shown on the left. A sample HTML syntax that shows alternate forms of the same document and the QoS required to retrieve each form of the data is shown above. The actual QoS specification is based on the Fore SPANS signaling API.

to use the OS/network interfaces to request resource reservations in order to achieve desired QoS. Figure 1 shows the proposed QoS extensions to URL and Socket classes in Java. In addition to the standard attributes such as protocol id, port number, and file name, the new URL class includes an optional QoS attribute that allows a client to specify its delay and bandwidth requirements. Some of the QoS parameters such as the bandwidth and packet burst sizes can be automatically derived using media-specific profiles such as the Real-Time Transport Protocol (RTP) payload profiles. HTTP protocol handler adds the QoS attribute to its `GetWithQoS` request. In addition, we have extended the BSD socket interface with a new `setsockopt(socket, IPPROTO_IP, SET_QOS, QoSSpec)` option that allows resource reservations using both RSVP and TCP/UDP protocols. Similar extensions have been proposed in [16].

3 Provision for alternate forms of content to accommodate heterogeneous networks

Recently, two new elements, namely, **embed** and **insert** elements, have been proposed to be added to HTML for supporting overlays[8] and are very useful for supporting alternate forms of content. However, the syntax needs to be further extended to describe choices of multiple formats and to specify hierarchical encoding of embedded objects. We propose a new element called **altdata** within an *insert* element for this purpose.

Figure 2 shows an example syntax for a document called "newsclip". A browser has a choice of retrieving alternate data formats indicated by *altdata elements*. For instance, `clipl1.mpg` offers a high-quality (`rchlevel=1`), 30 frames-per-second MPEG clip whereas `clip.h263` offers a low bit-rate video using H.263 encoding. In addition, two versions in still image format (a large-size GIF file vs a thumbnail-size GIF file) are also offered.

In the case of the MPEG version, the video itself is encoded using a multi-layer, hierarchical encoding and individual layers are further specified using the *overlay* elements. Thus, an application can use the QoS requirements (specified by attributes such as `qos_peak` for MPEG video or `qos_fromlength` attribute in case of GIF files) to choose an appropriate form of the content. Note that structural attributes `id` and `pid` help in linking the various overlays to the appropriate `altdata` element of a component.

The example for a H.263-based video clip shows another option. In this case, the QoS information is stored in an external file. The purpose of this option is to avoid requiring an author of the HTML document to specify QoS attributes and instead allow automatic derivation of QoS attributes from a media-specific profile such as an RTP payload profile for a H.263 codec.

4 OS Support for Real-Time, Adaptive Threads in Java

Java applets that would perform real-time animations or transfer and play back multimedia streams pose a special challenge to Java programmers. Because the applets should be written without any assumptions about the capability and availability of host resources (CPU, memory, disk/bus bandwidth, network interface

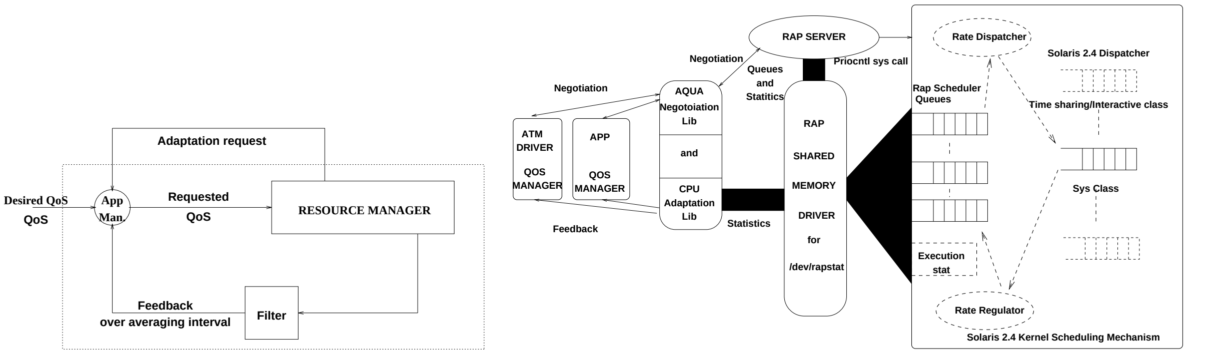


Figure 3: Figure on left shows the feedback-based AQUA framework for resource management. Figure on right shows the organization of the RAP CPU scheduler. A RAP server in user space cooperates with applications using the AQUA QoS negotiation library and a RAP-specific adaptation library.

capacity), the applets must be designed to adapt to maximize the use of available resources and must also be able to query and negotiate predictable QoS with the host OS. For instance, a video player applet can neither (a priori) specify the amount of computing time necessary to play back a video frame nor assume a certain playback rate (e.g., 15 frames per second) without any knowledge of the host capabilities.

Our goal is to introduce the notion of real-time execution in the Java run-time environment and define an OS interface for dynamic adaptation and negotiation of predictable QoS with a host OS[4, 11]. Our proposal is based on the AQUA (Adaptive QUality of service Architecture)[11] architecture that has been implemented in the Sun Solaris 2.4 operating system. In the following, we first present an overview of AQUA and then describe extensions to the Java threads.

4.1 An Overview of Resource management in AQUA

Figure 3 depicts the AQUA framework for managing host resources[11]. The framework is based on a cooperative model in which resource managers and applications negotiate and adapt to deliver predictable QoS. Each resource is managed by a resource manager that performs admission control on QoS requests for the resource. An application-level QoS manager requests desired QoS from the resource manager and adaptively re-negotiates the desired QoS based on the resource requirements and availability. The QoS Manager may also modify an application's behavior (such as changing the playback rate in a video player) when adequate resources are not available. We have applied the framework to the problem of managing a variety of host resources such as CPU time, network interface bandwidth, and display color map [9]. We will use here an example of CPU scheduling for the purpose of describing QoS management in Java threads.

The CPU management component of AQUA shown in Figure 3 is called RAP (Rate based Adjustable Priority scheduler)[18] and consists of the following parts: an application level QoS manager responsible for application level adaptation, a CPU server (called RAP server) that allocates the CPU resources among multiple applications, a QoS negotiation library that applications use to negotiate for resources with the RAP server, and a CPU-specific adaptation library that supports estimation of computing time needed on a CPU.

The RAP implementation separates policy from mechanism: CPU scheduling mechanism is built into the kernel while admission control and adaptation mechanisms are implemented by the RAP server in the user space (similar to microkernel approaches[17]). All processes communicate with the RAP server using the AQUA's negotiation library. The RAP adaptation library and the server share RAP kernel data structures and information on execution statistics using shared memory. Briefly, RAP scheduler works as follows:

- RAP maintains an estimate of the average computing time currently needed by each admitted process based on its recent CPU usage and desired rate of execution specified by the process. Based on these esti-

CPU QoS Class

```
public class RapQoS {
    /* lwp id */
    private int id;
    /* CPU id */
    private int rid;
    /* resource specific id */
    private int rspec_id;

    /* optional */
    /* set to zero */
    /* if not known */
    private int compute_sec;
    private int compute_msec;
    private double rate;
    private double minrate;
    /* acceptable jitter */
    private int jitter;

    /* adaptation */
    /* upgrade/degarade */
    private int adapt_actions;
    /* load to upgrage or degrade */
    private double load;

    /* methods to set values */
    /* and get values */
    ....
}
```

Adaptable Interface

```
package aqua.thread;
public interface Adaptable
    extends Runnable {
    public abstract
        int adapt(RapQoS rapqos);
}
```

RAP Scheduling Class

```
public class Rap implements Adaptable {
    Adaptable target;
    RapQoS rapqos;
    /* init RAP scheduling */
    public native int sched(RapQoS rapqos);
    /* exit from RAP scheduling */
    public native int sched_exit();
    public Rap(RapQoS rapqos) {
        sched(RapQoS rapqos)
    }
    public int adapt(RapQoS rapqos) {
        if (target != null) {
            return target.adapt(rapqos);
        }
        return -1;
    }
    ........
}
```

Example of Adaptive Real Time Applet

```
public class RTApplet extends
Applet implements Adaptable {
    Thread runner;
    Rap RealTime;
    RapQos RQoS;

    public void init() {
        runner = new Thread();
        Rqos = new RapQos(RATE,MINRATE,JITTER);
        RealTime = new Rap(Rqos);
    }

    public void adapt(RapQoS qos) {
        if(qos.getAction() == DEGRADE) {
            /* take what ever application */
            /* specfic action that is required */
        } else {
            /* take what ever application */
            /* specific action that is */
            /* needed to improve QoS */
        }
    }

    public void run() {
        /* work */
    }
}
```

Figure 4: Simplified samples of code that show real time extension to JAVA based on RAP scheduling. The RapQos class provides a means for applications to express CPU QoS. When applications cannot specify a particular parameter they can leave them blank.

mates, RAP's admission control algorithm in the server calculates the available capacity for the new process, determines whether the new process can be admitted, and if so, allocates the new process a computing time based on the remaining available capacity. The new process is assigned a priority based on its requested rate and is inserted in a priority-based queue for execution. The RAP scheduling mechanism in the kernel uses the parameters set by the server to schedule the processes appropriately.

• After the application starts executing, the average computing time it needs, and its rate of execution are monitored over an averaging interval. At the end of the averaging interval for a process, the RAP scheduler provides feedback to the application-level QoS manager about the observed rate of progress.

• Based on the feedback, the QoS manager can either change its QoS reservation request or adapt to available QoS. If the observed QoS is not within the acceptable range, the application-level QoS manager may then decide to adapt by reducing either the desired rate of execution or the amount of computing time needed per execution. For instance, the latter can be achieved by selectively playing back only some layers of a hierarchically encoded video stream.

• When resource availability changes due to admission of new processes or increase in the CPU usage by previously admitted processes, the RAP server handles such overload cases based on a user-specified policy. For instance, one policy may dictate that all the admitted processes (or processes in a particular application-level class such as audio or video) be asked to reduce their individual CPU requirements by a fraction proportional to their share of CPU capacity.

4.2 Extending Java to support real time threads

To enable Java threads to negotiate necessary QoS and adapt to available resources on a host, two modifications should be made to the Java threads library. First, Java Runnable classes must be extended to include necessary interfaces for resource estimation, QoS negotiation, and adaptation. Second, Java threads must also be mapped into RAP kernel threads to take advantage of the AQUA framework. The latter part depends on the kernel threads supported by a particular OS.

We have added a new interface called `Adaptable` that extends the Java Runnable classes. As shown in Figure 4, the interface exports an `adapt` method that can be called by the CPU QoS manager to initiate adaptation. A new class (called RapQoS) allows an application to specify its CPU requirements such as desired rate of execution and amount of computing time needed per execution (the latter can be omitted as explained earlier and the application is admitted based on a default estimate and is subject to adaptation if the actual CPU time requirement exceeds the available CPU capacity). For example, the sample application in 4 only specifies minimum and maximum bounds on rate of execution and acceptable rate jitter.

The new Rap class takes an existing Java thread and converts it into a real time, adaptive thread. The class RTApplet adds adaptability to an applet by exporting an `adapt` function called by RAP when necessary. The adapt function would typically perform actions such as changing playback rate or changing the network QoS reservation to match the new playback rate.

We have implemented the new classes in Java and our current prototype simply converts the entire Java process into a single, real-time AQUA process. We are currently porting the Java threads package to use RAP kernel threads.

5 QOS support at the transport level

Extensions to API and HTTP to allow specification of QoS requirements must also be accompanied by appropriate support in protocol stack. For example, the transport protocols (TCP and UDP) must map these QoS requirements onto appropriate network-level resource reservations.

Most of the present approaches for mapping IP over ATM [12] involve transparent, on-demand establishment of ATM virtual circuits to a destination host with no explicit input on QoS requirements from applications. Our experiments over an ATM testbed consisting of five ATM switches and results reported by others[2] indicate that end-to-end TCP and UDP performance can suffer severe degradation under this approach. Early Packet Discard[14] and per-VC packet queuing improve TCP performance, but fail to ensure that a particular TCP/UDP flow will receive a desired fraction of available bandwidth.

We have taken a different approach and extended the Solaris TCP/IP stream drivers and network interface drivers to allow dynamic discovery of the kind of QoS support (best effort vs ATM vs RSVP-based reservations) available over an outgoing network interface (FDDI, ATM, ethernet) to a destination IP address. Given a QoS request, the transport protocol maps the requested QoS onto an appropriate flow establishment. For instance, in a native ATM environment, a separate VC with appropriate QoS is established for a TCP connection in response to a *setsockopt* request[1]. The details of our implementation and results of a performance evaluation are presented elsewhere [19].

6 Dynamic QoS Renegotiation for persistent HTTP connections

HTTP makes inefficient use of network resources by creating a new TCP connection for each request[13] and modifications to HTTP have been proposed to transport multiple requests over a persistent HTTP connection. However, use of a persistent connection poses an interesting new problem. The QoS requirements of each request would vary depending on the kind of document to be transferred. Moreover, bandwidth requirements of a single request (such as transfer of a MPEG video) can vary dynamically within a transfer. Thus, an ability to dynamically re-negotiate the QoS reservation of an ongoing connection is necessary. In addition, a Web client using RTP[15] for transfer of an audio or video stream receives feedback from its associated control protocol (RTCP) about the current end-to-end performance in terms of observed delay and losses. Such feedback can (and should) be used to change the QoS reservations to improve the playback quality of an ongoing transfer.

We have extended the ATM signaling framework to allow dynamic renegotiation of QOS during a connection's lifetime and extended the Unix *setsockopt* option to let an HTTP client renegotiate QoS reservations over a persistent HTTP (TCP) connection. We have incorporated our extensions into a prototype implementation over a network of five Fore ATM switches and Sun Sparc workstations and used the prototype to evaluate the benefits of dynamic QoS renegotiation. More details on our signaling extensions and experimental results can be found in [10].

[1] Currently, we have focused only on exploiting ATM functionality for adding QoS capability to the Web-based applications due to lack of widespread availability of RSVP support in Internet routers. However, our proposal should be equally applicable to RSVP-based reservations.

7 Summary

To enable Web-based applications and applets to take advantage of the QoS guarantees supported by host operating systems and underlying networking protocols, we have suggested a variety of enhancements and extensions to the existing Web infrastructure. These proposals include addition of the notion of QoS to the HTML, HTTP, and Java syntax, QoS negotiation and adaptation extensions to Java classes and threads, QoS reservation support at the socket and transport protocol layers, and ATM signaling extensions for dynamic QoS re-negotiattion for persistent HTPP connections. We have implemented many of our proposals in an experimental testbed consisting of a network of Sun workstations interconnected by five ATM switches and ongoing work includes implementation of HTML and HTTP extensions.

8 Acknowledgements

Many thanks to Raphael Finkel, Jim Griffeon and Brent Seales for their constructive comments. Thanks to FORE systems for providing source code for both ATM device drivers and signaling software.

References

[1] M. Buddhikot, G. Parulkar, and R. Goplakrishnan. Scalable Multimedia-On-Demand via World-Wide-Web (WWW) with QoS Guarantees. In *Proc. 6th Intl. Workshop on Network and Operating System Support for Digital Audio and Video(NOSSDAV), Zushi, Japan*, April 1996.

[2] Douglas E. Comer and John C. Lin. TCP Buffering And Performance Over An ATM Network. *Journal of Internetworking: Research and Experience*, 6(1):1–13, March 1995.

[3] G. Coulson, A. Campbell, P. Robin, G. Blair, M. Papathomous, and D. Shepard. The Design of a QoS-Controlled ATM-Based Communications System in Chorus. *IEEE JSAC*, 13(4):686–699, May 1995.

[4] Q-Thread: A New Execution Model for Dynamic QOS control of Continous-Media Processing. Kiyokuni kawachiya and hideyuki tokuda. In *6th International NOSSDAV Workshop*, April 1996.

[5] Raman Gopalakrishna and Guru Parulkar. Real-Time Signals: A Mechanism to provide Real-Time Processing Guarantees. Technical Report WUCS-95-06, Washington Univeristy- Dept. of Computer Science, March 1995.

[6] Ramesh Govindan and David P. Anderson. Scheduling and IPC mechanisms for continuous media. In *Proceedings of 13th ACM Symposium on Operating Systems Principles*, pages 68–80, October 1991.

[7] JavaSoft Inc. Java. Technical Report http://java.sun.com, 1996.

[8] C Kindel et al. Inserting multimedia objects into HTML3. *W3C-Draft*, December 1995.

[9] K Lakshman. *AQUA: Aqdaptive Quality of Service Architecture*. PhD thesis, Univeristy of Kentucky, 1996. In preparation.

[10] K. Lakshman and R. Yavatkar. An Empirical Evaluation of Adaptive QOS Renegotation in an ATM Network. In *Proc. 6th Intl. NOSSDAV Workshop, Zushi, Japan*, April 1996.

[11] K. Lakshman and R. Yavatkar. AQUA: An Adaptive End-System Quality of Service Architecture. In W. Effelsberg, O. Spaniol, A. Danthine, and D. Ferrari, editors, *High-Speed Networking for Multimedia Applications*. Kluwer Academic, 1996.

[12] M. Laubach. Classical IP and ARP over ATM. *Internet RFC-1577*, Januray 1994.

[13] Jeffrey C. Mogul. The Case for Persistent-Connection HTTP. In *Proceedings of SIGCOMM'95*, August 1995.

[14] A Romanow and S Floyd. The Dynamics of TCP Traffic over ATM Networks. *IEEE JSAC*, May 1995.

[15] H. Schulzrinne and S. Casner. RTP: A Transport Protcol for Real-Time Applications. *IETF RFC1889*, 1996.

[16] Stardust Technologies. Windows Sockets 2 API, Revision 2.1.0. Technical report, January 1996.

[17] Hideyuki Tokuda and et al. Real-Time Mach: Towards Predictable Real-Time System. In *Proc. USENIX Mach Workshop*, pages 73–82, 1993.

[18] R. Yavatkar and K. Lakshman. A CPU Scheduling Algorithm for Continuous Media Applications. In *Proc. 5th Intl. NOSSDAV Workshop, Durham, New Hampshire*, pages 223–226, April 1995.

[19] R. Yavatkar, M. Manoharan, and K. Lakshman. QoS support for TCP/UDP and IP over ATM. Technical report, University of Kentucky, July 1996.