Application-Level QoS Control for Video-on-Demand

Using mobile agent technology, the ubiQoS middleware supports QoS tailoring and adaptation of video-on-demand flows in response to user preferences and terminal properties.

Paolo Bellavista and Antonio Corradi
Università di Bologna, Italy

Cesare Stefanelli
Università di Ferrara, Italy

As users continue to access the Internet from a growing set of heterogeneous access devices, they demand Web services tailored to their personal preferences and usage type (business or personal, for example). Client Internet access devices range from traditional workstations and PCs to laptops, personal digital assistants, and smart phones, and connectivity can be wired or wireless and continuous or intermittent. User requirements and device heterogeneity call for Web services with differentiated quality of service (QoS). In particular, services with response-time constraints, such as video-on-demand (VoD), require the differentiation, control, and dynamic adaptation of QoS.

Service providers and network operators need methodologies and mechanisms for managing runtime QoS. Although several recently proposed ad hoc protocols at the network layer have proven effective in limited networks,[1] they are incompatible with the Internet's best-effort model. Their implementation requires intermediate routers traversed by service packet flows, which will likely incur a long process of acceptance and diffusion. In addition, the protocols work at an abstraction level that complicates the embedding of functions such as application-specific adaptation and secure billing.[2]

The design, implementation, and deployment of QoS-aware Internet services can significantly benefit from a middleware approach at the application level.[3,4] (See the "Related Work on QoS for Internet Video-on-Demand" sidebar.) In VoD services, the middleware should exhibit several enabling properties:

- *QoS awareness*, to manage service components according to agreed-on QoS levels;
- *Location awareness*, to enable runtime decisions based on network topology and the current positions of involved resources;
- *Domain-specific adaptation*, to

Related Work on QoS for Internet Video-on-Demand

The emergence of commercial software systems, such as Akamai (www.akamai.com) and Inktomi (www.inktomi.com), that support the delivery of high-quality multimedia content to a large community of clients demonstrates the growing interest in Internet VoD distribution. Such systems exploit networks of statically installed servers and transparently load-balance requests among suitable servers near the client. In addition, several researchers have successfully designed and implemented distributed middleware for QoS provisioning (TAO[1] and MidART,[2] for example). These systems focus mainly on providing APIs for end-to-end QoS management and proposing mechanisms for QoS-aware resource allocation, scheduling, and control, especially in real-time systems. However, they don't deal with the support infrastructure's autonomous evolution in response to dynamic variations of request locations or with the automatic profile-based tailoring and adaptation of the provided QoS.

In the QoS-enabled VoD domain, several research projects propose interposing a proxy (also called a gateway or portal) between the client and server to provide distributed caching and adaptive transcoding of multimedia flows.[3-4] These projects focus on algorithms for efficient downscaling, adaptive bandwidth allocation, and content-based naming resolution, and do not consider dynamic proxy allocation or movement.

Of these projects, Reflector's active infrastructure for QoS tailoring and adaptation most resembles ubiQoS.[5] Reflector dynamically determines paths of multiple intermediate nodes between clients and servers. It aims to test the feasibility of distributing low-to-medium-bandwidth flows to thousands of Internet users simultaneously, and had significant success in broadcasting the NASA Pathfinder mission live. To use Reflector, however, you must preinstall its server infrastructure in proper points on the distribution network, and you cannot move middleware components or extend them with new functionality at runtime.

As in ubiQoS, recent research in active networks and services concentrates on forms of code mobility for implementing dynamic support infrastructures in VoD.[6] However, most current proposals do not let the execution state migrate with the code, which facilitates autonomous QoS negotiation and adaptation and active path rearrangement. In fact, after wide-scale deployment experiences, the Reflector developers noted that a major weakness of their system was its insufficient support for dynamic component distribution, reconfiguration, and extension. They indicated that code mobility is the most promising technology for a Reflector reimplementation.[5]

References

1. I. Pyarali, D.C. Schmidt, and R.K. Cytron, "Achieving End-to-end Predictability in the TAO Real-time Corba ORB," *Proc. 8th IEEE Int'l Symp. Real-Time and Embedded Technology and Applications*, IEEE CS Press, 2002, pp. 13-22.
2. O. Gonzalez et al., "Implementation and Performance of MidART," *Proc. IEEE Workshop on Middleware for Distributed Real-Time Systems and Services*, IEEE CS Press, 1997.
3. A. Fox et al., "Adapting to Network and Client Variation Using Infrastructural Proxies: Lessons and Perspectives," *IEEE Personal Comm.*, vol. 5, no. 4, Aug. 1998, pp. 10-19.
4. C.D. Cranor et al., "Enhanced Streaming Services in a Content Distribution Network," *IEEE Internet Computing*, vol. 5, no. 4, July 2001, pp. 66-75.
5. F. Kon, R.H. Campbell, and K. Nahrstedt, "Using Dynamic Configuration to Manage a Scalable Multimedia Distribution System," *Computer Comm.*, vol. 24, no. 1, Jan. 2001, pp. 105-123.
6. B. Krupczak, K.L. Calvert, and M.H. Ammar, "Implementing Communication Protocols in Java," *IEEE Comm.*, vol. 36, no. 10, Oct. 1998, pp. 93-99.

match service distribution to dynamic changes in the provisioning environment (through format transcoding and media resynchronization, for example).

These guidelines, discussed in greater detail in the "Middleware Design Guidelines" sidebar, next page, drove the design and implementation of ubiQoS, our application-level middleware for QoS tailoring and adaptation of VoD flows over standard IP networks. Unlike commercially available multimedia distribution systems, ubiQoS automatically installs its middleware components where and when needed and dynamically migrates them to rearrange the VoD flow paths at provision time. UbiQoS components autonomously and locally control network resources, and promptly perform adaptation operations near dynamically determined critical points in the infrastructure. Autonomy and locality of resource- and service-management operations are crucial to achieving scalability in the open and global Internet environment. To the best of our knowledge, ubiQoS is the first such middleware implemented using Java-based mobile agents to exploit the mobility of both code and state intrinsic to the mobile-agent programming paradigm. This article presents the main design choices behind ubiQoS. Other implementation details are available at the ubiQoS Web site (http://lia.deis.unibo.it/Research/ubiQoS) and elsewhere.[5]

UbiQoS Basics

Users expect middleware to provide ubiquitous access to Internet-based VoD services. UbiQoS

Middleware Design Guidelines

Diffusing VoD services over the Internet largely depends on the middleware's ability to support the tailoring and adaptation of QoS levels.

- *Service tailoring* is the initial configuration phase in negotiating the proper QoS level for the service session, and accounts for differentiated user requirements, heterogeneous access devices and points of attachment, and available network resources. For example, portable devices generally require downscaled services that fit their limited hardware resources.
- *Service adaptation* involves tuning QoS in response to changes in resource availability at provision time and requires monitoring support to detect network and system conditions. If client devices include wireless terminals, runtime changes in resource availability are common.

Tailoring and adapting Internet VoD can significantly benefit from open, portable, and extensible middleware with full visibility of network, system, and service properties.

Application-level middleware solutions are suitable for best-effort Internet service. In fact, tailoring and adaptation operations require the infrastructure to closely interact with application-layer service components. For example, standardized solutions and protocols for directory and discovery at the application level can improve the dynamic retrieval of user-preference and terminal-capability information. Domain-specific adaptation requires operations (such as format transcoding) and flow characteristic modifications (such as frame rate, image resolution, and image size), again suited to the application level.[1,2] Security management also benefits from working at the application level. For example, service admission control should depend on system resource state, requested QoS level, and user location; accounting should associate users with provided services in a nonrepudiable way and should bill them for their resource consumption.

In addition, any middleware for the global Internet should address scalability and openness issues. Thus, we need decentralized solutions with middleware components autonomously performing corrective control and adaptation operations overcoming the limitations of traditional client-server QoS management.[3] Novel middleware components should have full visibility of involved network resources and should operate on them locally to ensure prompt reactions. Emerging programming paradigms based on code mobility let middleware implement mobile components that migrate along the VoD distribution path at negotiation time and

continued on p. 19

offers two dimensions of ubiquity:

- *Accessibility*. UbiQoS automatically tailors and adapts VoD flows' QoS levels to fit user preferences, access devices, and available network resources, letting users receive VoD flows anywhere.
- *Middleware*. UbiQoS components autonomously scatter among network hosts along the paths from VoD servers and clients, depending on the emergence of congestion points.

UbiQoS supports VoD service tailoring at negotiation time and VoD service adaptation at provision time.

When a user requests VoD content, ubiQoS retrieves user preferences and current device capabilities, expressed as profiles stored in lightweight directory access protocol (LDAP) servers. Using its discovery service, ubiQoS searches for a server that has the requested VoD content with a QoS level greater than or equal to that expressed in the profiles. After identifying a suitable server, ubiQoS establishes a server-to-client network path for the VoD flow. A set of dynamically installed ubiQoS components negotiates the QoS on any segment along this path and decides which downscaling operations to perform at which nodes. UbiQoS components also perform application-level admission control and reserve local resources. They accept requests for new VoD flows (or for enhancing the QoS level of established flows) only if enough local resources are available at request time.

The state of system and network resources along the distribution paths determines how ubiQoS will adapt the QoS of VoD flows. An ad hoc monitoring component controls resources' state during service provision and triggers adaptation operations to adjust the QoS level if resources change. Adaptation can affect the transmitted VoD data (from transcoding to frame resizing and from merging or splitting multilayered tracks to reducing frame resolution and rate), eventually modifying the established VoD path. When this occurs, ubiQoS triggers a new negotiation phase and distributes new middleware components where needed. The type of corrective operation depends on the client's user and terminal profiles, which assign priorities to available adaptation modes. For instance, if the network is congested, one user might prefer to receive frames with a lower color depth whereas another might prefer a lower frame rate.

Middleware Design Guidelines (continued)

continued from p. 18

rearrange themselves in response to modifications in network and system conditions at provision time.

At negotiation time, dynamic middleware-distribution components simplify and enhance QoS tailoring via two strategies:

- *Resource admission control and reservation*. A service request's admission should depend on the current resource state — for example, the list of accepted VoD flows with the corresponding resource reservation — and user authorizations. Reservation and authorization operations benefit from collocation with the admission-controlled resource.
- *Rapid protocol prototyping and deployment*. Mobile middleware components permit the dynamic installation of new application-level protocols whenever the protocols are unsupported or their applicability is too limited to motivate their introduction at the network level.

At provision time, mobile code-based middleware components facilitate the support of QoS adaptation via both mechanisms and strategies:

- *Local monitoring of current QoS*. Centralized management solutions create bottlenecks in critical situations and overload network resources near congestion points — a micromanagement problem. Delegated management, on the contrary, lets middleware distribute components operating autonomously and locally to the bottlenecks, even if the network is temporarily disconnected.[3]
- *VoD path rearrangement*. Mobile middleware components can change the provision path in response to congestions and failures in service providers, intermediate hosts, or network links. In case of unacceptable QoS degradation, they can determine an alternative path, migrate to the new intermediate hosts, and resume the VoD session.

The combination of these mechanisms and strategies lets middleware effectively provision VoD services to any heterogeneous access device with the desired QoS level.

References

1. A.T. Campbell, "QoS-Aware Middleware for Mobile Multimedia Communications," *Multimedia Tools and Applications*, vol. 7, no. 1–2, 1998, pp. 67–82.
2. E. Amir, S. McCanne, and R. Katz, "An Active Service Framework and Its Application to Real-time Multimedia Transcoding," *Computer Comm. Rev.*, vol. 28, no. 4, Oct. 1998, pp. 178–189.
3. P. Bellavista, A. Corradi, and C. Stefanelli, "An Integrated Management Environment for Network Resources and Services," *IEEE J. Selected Areas in Comm.*, vol. 18, no. 5, May 2000, pp. 676–685.

Adopted Technologies

UbiQoS exploits emerging and accepted application-level technologies to facilitate its use over the Internet. In particular, it adopts the real-time transport protocol (RTP)[6] to transmit multimedia packets; the Java media framework (JMF, http://java.sun.com/products/java-media/jmf) to process VoD flows; and the W3C's composite capabilities/preference profile (CC/PP, www.w3.org/Mobile/CCPP), an emerging standard based on the Resource Description Framework (RDF), to represent user and terminal profiles.

RTP is the most widely used application-level protocol for QoS control over best-effort networks. In addition to transmitting data, RTP requires communication end points to exchange QoS-related monitoring information contained in sender and receiver reports. Sender reports include monitoring information such as RTP time stamps and the number of transmitted packets or bytes. Receiver reports contain statistics about the received flows, such as interarrival jitter and fraction of lost packets since the last report.

JMF is Sun's portable framework for acquiring, elaborating, and visualizing multimedia flows. It provides components for encapsulating VoD sources and receivers — for example, acquisition hardware, visualization and encoding software, and incoming and outgoing network flows — and operating transformations such as compressions and format transcoding. With regard to multimedia flow transport and control, JMF exploits RTP and propagates the visibility of sender and receiver reports to application-level service components.

The differentiation of user requirements and the wide heterogeneity of client devices require well-represented and maintained user and terminal profiles. These should include user preference information (desktop interface information, required security level, and subscribed services and corresponding QoS requirements, for example) and terminal characteristics (display size, available memory, operating system, and so on).

Middleware Components

The ubiQoS infrastructure consists of several components that dynamically migrate to hosts along the VoD distribution path to tailor and adapt QoS levels.

- *Proxies* migrate where needed according to client location at provision time. They move throughout the network, composing a dynamically determined active path between client and server, and then remain in place to serve

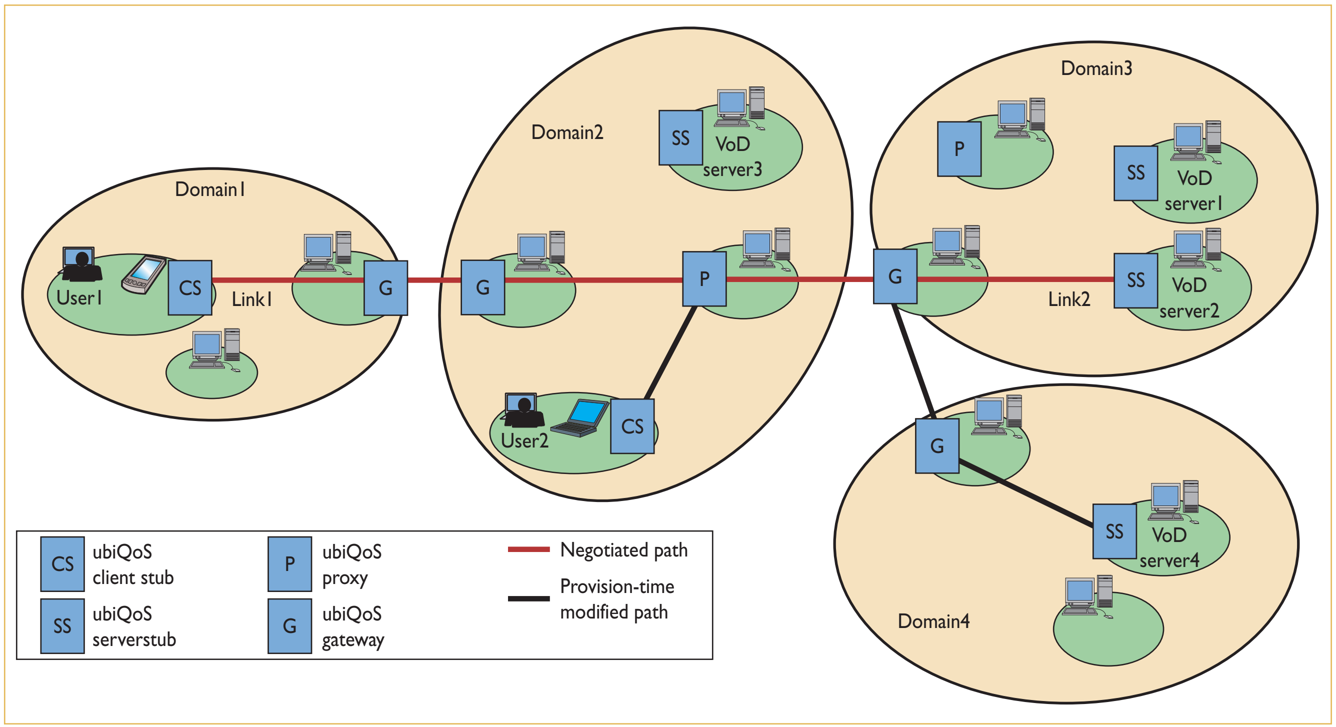

Figure 1. UbiQoS tailoring and adaptation operations. The proxy adapts the VoD transmission to provide the same VoD content to user1 and user2 with different QoS levels.

successive requests. Proxies perform admission control and reservation for incoming and outgoing flows. They also monitor system- and application-level resources and trigger local QoS adaptation operations. They coordinate with adjacent ubiQoS components both in the initial negotiation phase and at provision time when variations in resource availability occur.

- *Gateways* extend ubiQoS proxies with additional naming and coordination functions. In fact, ubiQoS organizes clients, servers, and networked resources in hierarchies of locality abstractions, or *domains*, and exploits an articulated locality-aware naming service. A ubiQoS domain typically corresponds to a set of local area networks with common administration and management policies. Domains contain one ubiQoS gateway, possibly replicated with a master-slave solution. The gateway is the only component that can completely view neighbor domains and the ubiQoS components within them, which helps ubiQoS provide scalable naming solutions and make structure-management decisions.[2]
- *Stubs* integrate the ubiQoS infrastructure with legacy VoD servers and players. Client stubs transparently forward VoD requests to ubiQoS components and RTP flows to local visualization tools. Server stubs answer service requests from ubiQoS components by transparently encapsulating server VoD flows into RTP flows. Thus far, we have implemented client stubs for JMF and Mbone videoconferencing tools and server stubs for JMF data sources. (See www-mice.cs.ucl.ac.uk/multimedia.)

Figure 1 depicts a deployment scenario in which user1 and user2 (and their access devices) require a high-quality flow from the VoD server that must be downscaled at the ubiQoS proxy in domain2. If link1's bandwidth degrades at provision time, the proxy (after coordinating ubiQoS components in domain1 and domain2) will adapt the VoD transmission for the user by reducing the frame resolution or rate according to the receiver preference profile.

If available resources cannot meet the negotiated QoS — for example, if link2 fails — the gateway eventually establishes a new path segment. The gateway in domain3 launches a discovery for a suitable VoD server in nearby domains, then negotiates with new ubiQoS-enabled hosts, and finally restarts the flow transmission from the point at which it was interrupted (if server4 supports random access to the VoD content). Apart from the time interval to establish the new path, the server swap is transparent to receivers and other intermediate ubiQoS components.

Gateway Architecture

All ubiQoS middleware components have similar modular architectures, as Figure 2 shows. The gateway is the most complete component and includes all basic modules. Other components have more limited duties and include only a subset of the gateway functions. Proxies do not need LDAP or discovery servers, for example, and stubs provide functionality only for gateway discovery and RTP reception and transmission.

The QoS Manager module coordinates the other local modules and decides the QoS level to offer the next ubiQoS component in the VoD path. At negotiation time, it combines QoS requirements from user and terminal profiles and reservation data from the Admission Control module. Any VoD service usually allows a space of possible values for QoS parameters, such as frame rate, size, and resolution. The QoS Manager enforces a specific point in the admissible space by using QoS information and the local host resource consumption policy. UbiQoS currently provides two simple alternative policies:

- A *thrifty policy* enforces the QoS point that minimizes local resource consumption according to a specified function cost.
- A *greedy policy* chooses the best allowed QoS point by reserving the maximum amount of local resources.

At provision time, the QoS Manager can also move the chosen QoS point in the admissible range to maximize a cost function with weighted QoS parameters, depending on user and terminal profiles.[5] If bandwidth degrades, for example, a videoconference user could opt for a high-quality audio stream over video frame rate.

The QoS Manager commands the QoS Adaptation module, which works to maintain the negotiated QoS point. Adaptation exploits JMF multiplexers and demultiplexers, codecs, and renderers, together with Java-based transcoding components developed within the project. For instance, we've implemented a family of ad hoc codecs to convert several VoD input formats into MPEG flows with quality parameters specified at runtime (see the ubiQoS site for details). The Adaptation module also maintains local caches of served flows so it can respond promptly to incoming client requests with compatible service requirements using a least recently used-based replacement strategy. For any VoD flow, the adaptation module stores the version received with the best QoS according to its cost function.

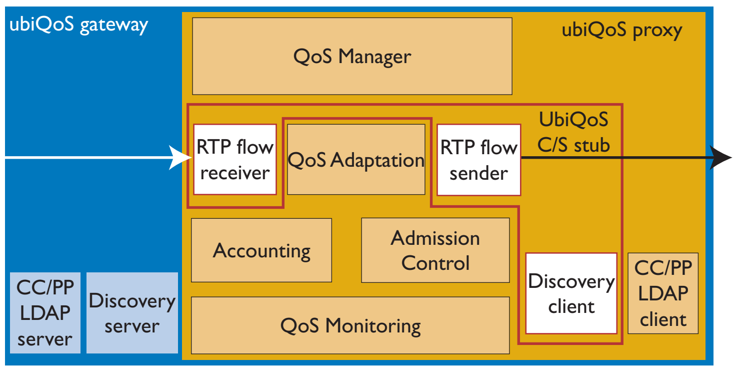

Figure 2. UbiQoS components' modular architecture. The most complete component, the gateway, comprises all basic modules. Proxies and stubs contain subsets of gateway functions.

The underlying modules provide the QoS Manager's basic functions. The QoS Monitoring module, for example, observes the system- and application-level states of resources and service components local to the ubiQoS proxies and gateways. The Admission Control module registers any currently served VoD flow traversing the local ubiQoS component and, depending on both the information from QoS Monitoring and the set of currently accepted flows, accepts or rejects the new service request. The Accounting module authenticates users and, using QoS Monitoring data, logs the QoS levels provided to subsequent ubiQoS proxies and gateways in the active paths. Accounting data is stored locally to the consumed resources and can be processed offline for billing or other purposes.

UbiQoS also exploits a flexible and scalable naming support system that integrates discovery and directory servers.[7] UbiQoS uses LDAP-compliant directory servers to store globally accessible information such as CC/PP profiles. Jini-based discovery lookup servers make the available VoD content and ubiQoS components accessible in any domain locality. UbiQoS gateways implement both directory and discovery server-side operations. Profiles are only partially replicated on gateways, and requests for nonlocal profiles are forwarded to neighbor gateways. Other ubiQoS components only implement lightweight client functionality and interrogate the modules for discovery or directory of the gateway in their locality.

Implementation

We built the ubiQoS middleware on top of the Java-based Secure and Open Mobile Agent platform for developing and deploying Internet services. SOMA provides APIs for mobility, communication, naming, monitoring, security, and interoperability.[2] In addition, SOMA models Internet localities with hierarchies of abstractions that fit the ubiQoS locality organization.[7]

Adopting a mobile-agent-based implementation technology for ubiQoS proxies and gateways offers two advantages:

- Code mobility lets ubiQoS components move where needed at provision time and dynamically update or extend their functionality without suspending service provisioning. For instance, the QoS Adaptation module can load new adaptation functions and caching strategies by exploiting code mobility and SOMA's distributed code repository.[2]
- UbiQoS's mobile-agent-specific state-migration property — its ability to move mobile agents together with their current execution state — lets ubiQoS proxies and gateways maintain both the user-specific QoS requirements and the QoS characteristics of previous segments in the active path. For example, if a link fails and a gateway dynamically establishes a new path segment, as Figure 1 shows, the ubiQoS gateway migrating to domain4 carries information about the QoS level in the domain2-to-domain3 segment as well as user2's QoS adaptation requirements, such as the user-specific cost function for the QoS Manager.

The ubiQoS Web site has additional details about the mobile-agent-based ubiQoS active path determination.

Our default deployment strategy is to install one gateway at each domain along the dynamically determined active paths. Other ubiQoS components move at runtime, depending on both the distribution tree of the served VoD flows and the resource availability along the paths. UbiQoS proxies move to where resource bottlenecks emerge dynamically, performing locally effective and prompt tailoring and adaptation operations on congested resources. Bottlenecks usually occur near network discontinuities, such as a move from a 622-Mbps ATM-based network to a 56-Kbps modem link, and in host capabilities, for example, locally to WAP gateways providing Web content to mobile phones.

Java in Internet VoD

Java plays a key role in Internet service provisioning because of its portability, object-orientation, dynamic class loading, and language-level security support.[8] The use of Java in Internet VoD has two drawbacks, however.

The Java virtual machine (JVM) hides system properties requested by the QoS Monitoring module. This module not only extracts the monitoring information available in real-time control protocol (RTCP) sender and receiver reports, but also provides the visibility of both system-level indicators (for example, CPU load and packet collision rate) and application-level indicators (such as method invocation and memory allocation for any Java active thread). To achieve this kind of visibility, ubiQoS Monitoring integrates with platform-dependent monitoring mechanisms using the JVM Profiler Interface and Java Native Interface (JNI) (additional details are at http://lia.deis.unibo.it/Research/MAPI).

Moreover, because it is based on an interpreter approach, Java is often unsuitable for time-constrained operations such as on-the-fly multimedia format transcoding. Our experience with ubiQoS has shown that Java succeeds in online QoS adaptation of multimedia flows at typical Internet transmission rates. Java-based ubiQoS components are portable on any platform hosting the JVM; for improved performance, however, the QoS Adaptation module integrates with native plug-in codecs, such as JMF and the commercial Design MediaPalette (www.cinax.com/Products/mp.html), by exploiting standard JNI libraries containing platform-dependent code that is typically tied to specific targets. QoS Adaptation dynamically retrieves the list of installed plug-ins to bind only to locally available native components.

Experimental Results

To evaluate our approach's feasibility and effectiveness, we've deployed the ubiQoS infrastructure over several geographically distributed networks interconnected via GARR (the Italian Academic and Research Network), and using very different bandwidths. Each network locality is modeled by one SOMA domain and includes heterogeneous nodes (Sun Solaris workstations, Intel Pentium PCs with Microsoft Windows NT, and SuSE Linux).

In ubiQoS, the initial active path-establishment and negotiation phase involves the client, the dynamically retrieved server, and possibly some active intermediate nodes. Establishing an active path segment requires querying the domain discovery service, creating an RTP connection, reserving and controlling resources, negotiating the tailored QoS, and, when needed, migrating ubiQoS proxies and gateways.

Figure 3 shows the near-linear dependence of path setup time on the number of intermediate active nodes. The figure also shows the effect of the number of migrations on path setup time: when a service request triggers the installation of ubiQoS proxies and gateways over just a small set of new active nodes, the delay is significantly smaller.

In addition, the number of active nodes tends to be smaller than the number of traversed networks, even when they are distributed across the Internet. In fact, ubiQoS proxies and gateways need operate only where network bandwidth discontinuities or forking of the VoD distribution tree exist. In general, the path setup time is significantly longer than the time necessary to establish a single RTP connection between one client and one server, but it is acceptable for noninteractive VoD services because it affects only the initial delay before beginning visualization at the client side. Prompt adaptation reactions at provision time and significant network traffic reductions in multicast distribution counterbalance the initial path-setup overhead.

Because ubiQoS components autonomously monitor, control, and manage their local path segments, reaction time — measured as the time interval between congestion occurrence and the start of adaptation operations in the closest proxy or gateway — is almost independent of the distance between the client or server and the congested segment, at generally about 150 milliseconds. In addition, ubiQoS proxies and gateways integrate standard RTP report transmissions with event-triggered exchanges of monitoring information provided by QoS Monitoring, thus overcoming the RTP limit on the statically determined report frequency (see the IETF Internet draft “RTCP-based Feedback: Concepts and Message Timing Rules”).

When a server multicasts the same VoD flow, ubiQoS proxies and gateways ascertain whether several receivers are available within their served localities; they split VoD flows only when and where necessary and downscale the VoD quality depending on their distribution subtree’s maximum QoS requirements. We measured the overall traffic, defined as the total number of bytes exchanged between all pairs of adjacent nodes along the distribution tree, by considering a large set of client distributions. On average, this approach reduces the overall traffic more than three times than traditional solutions that do not support IP-multicast.

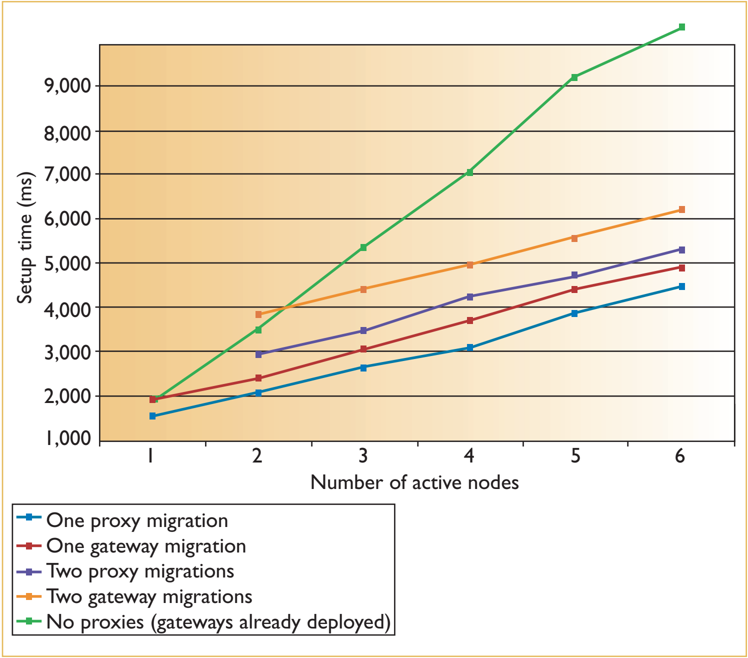

Figure 3. UbiQoS active path setup time. Path setup time varies with the number of intermediate active nodes.

Conclusive Remarks

Our experimental results show the feasibility of application-level middleware solutions based on code mobility for Internet VoD services with differentiated QoS, and are stimulating additional work. For example, we are extending ubiQoS to include VoD client stubs based on the Java 2 Micro Edition and VoD multimedia players such as TealMovie (www.tealpoint.com), targeted to wireless Palm devices with very limited visualization capabilities. In addition, we are considering the research area of peer-to-peer multimedia exchange, for instance, to use UbiQoS for interactive videoconference applications.

Acknowledgments

The Italian Ministry for Instruction, University, and Research (MIUR) and the Italian National Research Council (CNR) supported this work in the framework of the FIRB Web-Minds and IS-Manet projects.

References

1. X. Xipeng and L.M. Ni, “Internet QoS: a Big Picture,” *IEEE Network*, vol. 13, no. 2, Mar. 1999, pp. 8-18.
2. P. Bellavista, A. Corradi, and C. Stefanelli, “An Integrated Management Environment for Network Resources and Services,” *IEEE J. Selected Areas in Comm.*, vol. 18, no. 5,

May 2000, pp. 676-685.
3. R. Koster and T. Kramp, “Structuring QoS-Supporting Services with Smart Proxies,” *Proc. IFIP/ACM Int’l Conf. Distributed Systems Platforms* (Middleware 2000), LNCS 1795, Springer-Verlag, Apr. 2000, pp. 273-288.
4. D. Chalmers and M. Sloman, “A Survey of Quality of Service in Mobile Computing Environments,” *IEEE Comm. Surveys & Tutorials*, vol. 2, no. 2, 1999, pp. 2-10.
5. F. Baschieri, P. Bellavista, and A. Corradi, “Mobile Agents for QoS Tailoring, Control, and Adaptation over the Internet: The ubiQoS Video on Demand Service,” *Proc. Int’l Symp. Applications and the Internet* (SAINT’02), IEEE Computer Soc. Press, 2002, pp. 109-118.
6. H. Schulzrinne et al., “RTP: A Transport Protocol for Real-Time Applications,” Internet Eng. Task Force RFC 1889, Jan. 1996; www.ietf.org/rfc/rfc1889.txt.
7. P. Bellavista, A. Corradi, and C. Stefanelli, “Mobile Agent Middleware for Mobile Computing,” *Computer*, vol. 34, no. 3, Mar. 2001, pp. 73-81.
8. B. Krupczak, K.L. Calvert, and M.H. Ammar, “Implementing Communication Protocols in Java,” *IEEE Comm.*, vol. 36, no. 10, Oct. 1998, pp. 93-99.

Paolo Bellavista is a research associate of computer engineering at the University of Bologna. His interests include mobile agents, pervasive computing, systems/service management, location/context-aware services, and adaptive multimedia. He received a PhD in computer science engineering from the University of Bologna. He is a member of the IEEE, the ACM, and the Italian Association for Computing. Contact him at pbellavista@deis.unibo.it.

Antonio Corradi is a full professor of computer engineering at the University of Bologna. His research interests include distributed systems, object and agent systems, systems/service managements, and pervasive computing. He received an MS in electrical engineering from Cornell University. He is a member of the IEEE, the ACM, and the Italian Association for Computing. Contact him at acorradi@deis.unibo.it.

Cesare Stefanelli is an associate professor of computer engineering at the University of Ferrara. His research interests include distributed and mobile computing, mobile code, network and systems management, and network security. He received a PhD in computer science from the University of Bologna. He is a member of the IEEE and the Italian Association for Computing. Contact him at cstefanelli@ing.unife.it.