A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation

Ian Foster
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439, U.S.A.

Alain Roy
Department of Computer Science
The University of Chicago
Chicago, IL 60637, U.S.A.

Volker Sander
Central Institute for Applied Mathematics
Forschungszentrum Juelich GmbH
52425 Juelich, Germany

Abstract—**Reservation and adaptation are two well-known and effective techniques for enhancing the end-to-end performance of network applications. However, both techniques also have limitations, particularly when dealing with high-bandwidth, dynamic flows: fixed-capability reservations tend to be wasteful of resources and hinder graceful degradation in the face of congestion, while adaptive techniques fail when congestion becomes excessive. We propose an approach to quality of service (QoS) that overcomes these difficulties by combining features of reservations and adaptation. In this approach, a combination of online control interfaces for resource management, a sensor permitting online monitoring, and decision procedures embedded in resources enable a rich variety of dynamic feedback interactions between applications and resources. We describe a QoS architecture, GARA, that has been extended to support these mechanisms, and use three examples of application-level adaptive strategies to show how this framework can permit applications to adapt both their resource requests and behavior in response to online sensor information.**

I. Introduction

Network applications that need to achieve reliable end-to-end performance typically make use of either reservations or adaptation. When using reservations, applications usually specify quality of service (QoS) requirements when a connection is established and do not change them subsequently; the QoS system in turn guarantees that (modulo system failures or preemptions) the reservation will not be reduced during the lifetime of the application [1], [2]. In contrast, applications that use adaptation do not make reservations but instead adapt to the network conditions at hand by responding to some form of feedback, whether explicit (notification of network conditions) or implicit (noticing that bandwidth is low). Adaptation may occur when the application detects a problem or when the application is notified that a problem may exist [3], [4], [5], [6], [7].

Both reservations and adaptation have been proven effective in many situations, but also have significant limitations: particularly when dealing with high-end applications featuring high-bandwidth, dynamic flows. Fixed-capability reservations can waste bandwidth and do not permit graceful degradation in application performance when resource management policies mandate changes in allocations. Adaptive techniques inevitably fail when congestion reduces available resources below acceptable limits [8], [9].

Published in *Proceedings of the Eight International Workshop on Quality of Service (IWQOS 2000)*, pp. 181–188, June 2000.

In this paper, we describe an approach to QoS that combines features of both reservations and adaptation to address the difficulties just noted. At the core of this approach is a QoS architecture in which resources are enhanced with:

- *online control* interfaces that allow applications, or agents acting on their behalf, to modify resource characteristics (e.g., reservations) dynamically;
- *sensors* that allow applications (and agents) to detect when adaptation is required; and
- *decision procedures* that support the expression of a rich set of resource management policies.

These mechanisms in turn enable a wider range of application-level adaptation strategies than are supported in other architectures. For example, online control of reservations allows applications to request premium service when adaptive techniques fail to deliver; monitoring of reservations that change as a result of decision procedures embedded in resource managers allows for graceful degradation in application performance in response to preemption.

To explore these ideas, we have incorporated such mechanisms into a QoS architecture developed in previous work—the Globus Architecture for Reservation and Allocation (GARA) [10], [11]. We have completed a prototype implementation of this enhanced architecture, which has been deployed by ourselves and others on local and national testbeds.

We hypothesize that the mechanisms and associated control and information flows provided by this extended GARA architecture can be exploited to obtain more efficient resource usage than in purely reservation-based or application-based approaches, as applications can vary reservations and rates; to provide more flexible resource allocation strategies, as resources can change allocations over the course of a reservation; and to deliver more robust application performance, as applications can detect and respond to changes in allocations and resource state.

As a first step towards testing this hypothesis, we have used GARA mechanisms to implement three different adaptive strategies. The first two use a flow-specific packet loss sensor to adapt bandwidth requests to the QoS system in order to meet performance targets, for UDP and TCP flows, respectively; the third uses a sensor that provides information on changes in reservation level (as a result of preemption) to adapt transmission rate for bulk data transfer applications. In each case, we present novel decision procedures and demonstrate that we can deliver interesting adaptive behaviors via a combination of online monitoring and control.

In the rest of this paper, we review the QoS requirements of high-end applications, describe our enhanced GARA architecture, and present our three adaptive strategies and the experimental studies that we have performed to evaluate their effectiveness. We conclude with a brief discussion of related and future work.

II. Motivation: High-End Applications

We are interested in providing QoS mechanisms for *high-end network applications* [11], in which individual flows can have high bandwidth, from a few megabits per second (Mb/s) to many tens or hundreds of Mb/s; there may be complex mixes of flows, from low bandwidth to high bandwidth and from low latency to high latency; and flows may change their requirements dynamically throughout their lifetime.

Applications with these characteristics arise in such areas as distance visualization, analysis of petabyte-scale scientific databases, online control of scientific instrumentation, and teleimmersion [12]. For illustrative purposes, we examine a teleimmersion example in more detail. Consider two or more users at geographically separate locations who are exploring collaboratively a three-dimensional visualization of experimental data. As in other telecollaboration systems, we have a number of streams with fairly constant rate and low to moderate bandwidth: audio and video streams for communication, and jitter- and latency-sensitive streams for the tracking data indicating user movements in the virtual space. In addition, we have streams with higher bandwidth and often variable rates, used for visualization data and (in some cases) database updates. Visualization data is calculated from the data set, and a representation of it, perhaps a set of polygons for rendering, is transmitted [13]. The actual amount of data sent depends on both the data being visualized and user actions, which may include zooming and movement in space and time. Contention for shared resources such as disk and CPU can also affect the transmission rate.

These characteristics place substantial demands on both network infrastructure and applications. For example, consider a situation in which several teleimmersion sessions are in operation simultaneously, while other groups are concurrently attempting to perform high-speed bulk-data transfers over the same network infrastructure, perhaps to stage data required for an experiment later in the day. With today's protocols and services, no group would obtain acceptable service.

We believe that concerns such as these require that resource providers be able to specify and implement flexible resource allocation policies. For example, in the situation just noted, resource providers might allocate resources to different teleimmersion sessions and bulk-data transfers differentially. Teleimmersion session *A* might have priority, while sessions *B* and *C* would be guaranteed some minimum service. Bulk-data transfers *D* and *E* would have lowest instantaneous priority but would be guaranteed service in terms of another "terabytes per hour" metric.

We also believe that a policy-driven framework of this sort can be effective only if applications themselves are provided with the information and control flows required to detect and adapt to policy-driven changes in resource allocations. For example, a teleimmersion session could respond to reduced (increased) resource availability by reducing (increasing) video rates or introducing (eliminating) data compression to noncritical users, while a bulk-data transfer could reduce (increase) its sending rate. The architecture that we present in this paper enables these sorts of adaptation.

III. Reservation and Adaptation Combined

Effective adaptive control requires three distinct mechanisms. In the language of [14], these are

- *actuators* that permit online control, for example, of resource allocations or application behavior;
- *sensors* that permit monitoring, for example, of resource allocations or application behavior; and
- *decision procedures* that allow entities to respond to sensor information, by invoking actuators.

As illustrated in Figure 1, these three elements act in concert to achieve adaptive control. For example, a sensor might signal a nonzero loss rate associated with a flow at a router. A decision procedure in the associated application can then execute to determine whether to reduce the sending rate or, alternatively, generate a request to a resource manager to create (or increase) a reservation for that flow, hence invoking an actuator.

In this section, we first provide an overview of the GARA architecture and then explain how we have extended it to support these three mechanisms.

A. GARA Overview

The Globus Architecture for Reservation and Allocation provides advance reservations and end-to-end management for quality of service on different types of resources, including networks, CPUs, and disks [10], [11].

A GARA system comprises a number of *resource managers* that each implement reservation, control, and monitoring operations for a specific resource. Resource managers can and have been implemented for a variety of resource types, hence the use of the term "resource manager" rather than the more specific "bandwidth broker" favored in the networking literature [15]. Uniform interfaces allow applications to express QoS needs for different types of resources in similar ways, hence simplifying the development of end-to-end QoS management strategies. Mechanisms provided by the Globus toolkit are used for secure authentication and authorization of all requests to resource managers. An information service allows applications to discover resource properties such as current and future availability.

The work described in this article involves just a single type of resource manager, namely, one that uses differentiated ser-

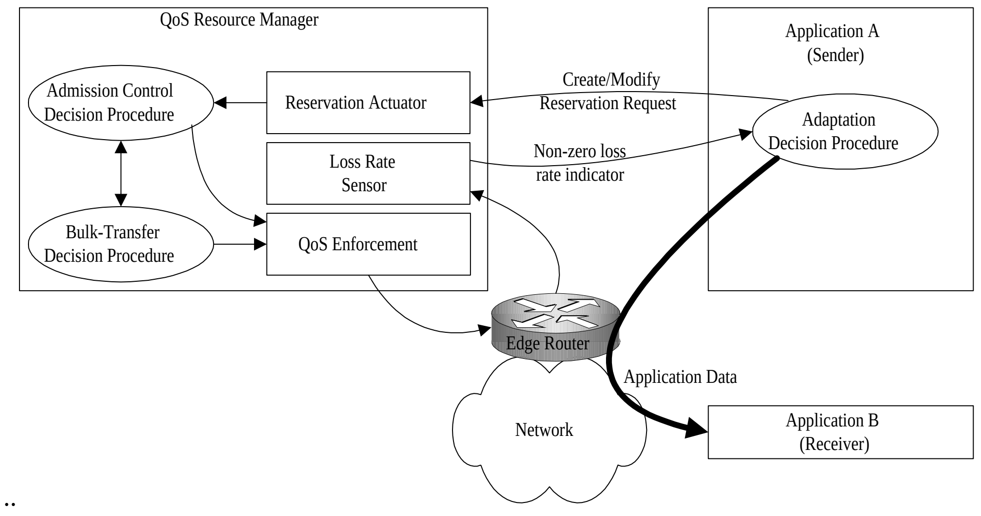
QoS Resource Manager
Admission Control Decision Procedure
Reservation Actuator
Loss Rate Sensor
Bulk-Transfer Decision Procedure
QoS Enforcement
Application A (Sender)
Create/Modify Reservation Request
Adaptation Decision Procedure
Non-zero loss rate indicator
Edge Router
Application Data
Network
Application B (Receiver)

Fig. 1. An example of how actuators, sensors, and decision procedures may be combined to provide adaptive control. We illustrate reservation adaptation in Application A, occuring as a result of a packet loss notification received from a router via the resource manager. The operation of this strategy is described in the text, as is the bulk transfer decision procedure that is also shown.

vices mechanisms [16] to implement network QoS. This resource manager uses the *expedited forwarding* per-hop behavior (PHB), as specified by the Internet Engineering Task Force's (IETF) Working Group on Differentiated Services, to provide a premium service. With careful admission control at the edge of the network, it is possible to build a network QoS system with reasonably strong bandwidth guarantees, even though traffic is treated as an aggregate in the core of the network.

The resource manager enables reservation requests (see below) by configuring the routers that it controls. In particular, it configures the ingress routers to classify, police, mark, and potentially shape, all packets that belong to a flow for which a reservation has been authorized, as is normally done for differentiated services. The expedited forwarding per-hop behavior drops packets that exceed the reservation, but allows small bursts of excess traffic using a token-bucket mechanism.

B. Actuators: Online Control

A first prerequisite for adaptation is support for online control of resource characteristics. (We are also interested in online control of application behavior, but that topic is beyond the scope of this article.) GARA supports this requirement directly via control functions that allow an application—or an agent acting on its behalf—to make and subsequently modify QoS reservations.

In the case of the network resources considered in this article, an application request to the resource manager specifies a start time and duration for the desired reservation; the IP addresses of the end hosts that will be communicating; the bandwidth required for the reservation; and the network protocol that will be used (TCP or UDP) [10]. Since reservations may be made in advance, not all information may be known at the time the reservation is made. In particular, an application may not know what port numbers will be used for communication until network communications begin. Therefore, GARA provides a "bind" operation, which simultaneously "claims" the reservation and provides this run-time information.

Both immediate and advance reservations are supported. Advance reservations simplify co-scheduling of scarce resources and help to ensure that resources are available for important events, such as scientific experiments.

GARA allows third parties to make, monitor, and modify reservations on behalf of an application. This capability allows us to separate adaptation logic from an application proper; in the case of advance reservations, it means that an application need not be running when a reservation is made. For brevity, we frame subsequent discussion as if only applications manipulate reservations; however, in practice, a third party can always be substituted.

C. Sensors

A second requirement for adaptive control is that we be able to determine the state of system components and detect state changes. This capability is provided via sensors associated with system entities to which other entities can subscribe, with notifications provided via some form of event service or callback mechanism. We have implemented two such sensors in our GARA prototype.

C.1 Loss rate sensor.

This sensor provides applications with information on packet loss rate in the network. This information can serve to indicate the application is either sending too fast or has an inadequate reservation.

We measure packet loss rates at the first hop router: that is, the router at which initial policing is performed by our differentiated services implementation. Our resource manager periodically queries this router, which because of its classification and policing role is able to provide statistics about the number of packets that have exceeded a flow's reservation.

The query to the router returns the number of packets that conformed to the reservation and were not dropped (p_c) and the number of packets that exceeded the reservation and were dropped (p_e), both of these quantities being since the last time the statistics were queried. If the resource manager detects a nonzero p_e value then it generates a callback to notify any subscribed processes that packet loss has occurred. This callback specifies both an estimated loss percentage and the currently unallocated bandwidth; an application might use the latter quantity as a guide when deciding whether to respond to a packet loss notification by attempting to increase its reservation vs. changing its behavior.

In computing the estimated bandwidth, we must deal with the complicating factor that the router uses a token bucket of size p_b to allow small bursts. The router updates its statistics only periodically (roughly every 10 seconds) and the resource manager cannot know if the token bucket was full or empty when the statistics were gathered. To avoid persistent underestimates of loss rates, we assume that the token bucket is at least half-full and reduce the number of conforming packets correspondingly. This adjustment is reflected in our formula for estimated fraction of packets that were dropped:

$$P = \frac{p_e}{p_c + p_b/2 + p_e}$$

We describe in Section IV how this sensor can be used to modify QoS reservations to meet application requirements, for both UDP and TCP flows.

C.2 Reservation change sensor.

Our second sensor is used to publish information about changes in resource allocations. The reason for these changes is described in the next subsection; here we note simply that we have a sensor capable of communicating such changes to interested entities.

D. Decision Procedures

The third component of an adaptive control architecture comprises the decision procedures that invoke actuators in response to sensor data.

In our environment, such decision procedures can occur in multiple locations. They clearly arise in applications, and indeed we give three such examples below. Decision procedures can also occur in resource managers; this can lead to interesting interactions.

Decision procedures may be invoked within a GARA resource manager at a number of points. Following authentication, an incoming request is first authorized and then executed. Decision procedures may be invoked at both stages: for example, to determine whether a request should be granted, in the first instance, and to reallocate resources in the second instance if the newly authorized reservation oversubscribes available resources.

To explore these ideas and demonstrate our ability to incorporate decision procedures in resource managers, we have implemented the following simple but highly effective procedure.

D.1 Bulk-data transfer procedure.

As noted above, bulk-data transfer (BDT) operations have service requirements expressible in terms of “terabytes per hour” rather than “Mb/s.” Satisfying such requirements in the face of congestion can require the use of premium service but need not always pre-empt other applications requiring premium service.

Our BDT decision procedure is designed to exploit this observation. In effect, it implements two classes of premium service, foreground and background, within a single premium service class. It does this by applying the following simple decision rules when processing requests to create, bind, or terminate reservations.

1. *Create foreground reservation*: Creation of a foreground reservation is authorized if at no time during the reservation period the sum of all foreground reservations would exceed the total available premium bandwidth.
2. *Bind foreground reservation*: Binding of a foreground reservation results in the requested bandwidth being allocated to the appropriate flow. If necessary, premium bandwidth is preempted from background flow(s), with callbacks being generated to notify interested parties.
3. *Cancel reservation*: The freed bandwidth is allocated to background flows with inadequate allocations, if any such exist, and callbacks are generated.
4. *Create background reservation*: Creation of a background reservation is always allowed.
5. *Bind background reservation*: Binding of a reservation results in a “fair share” of the unallocated premium bandwidth being allocated to the appropriate flow. (See below for a description of how this fair share is calculated.)

We describe below how an application can use the reservation change sensor triggered by this decision procedure to achieve sustained BDT rates without impeding foreground flows.

IV. Application-Level Adaptation Procedures

We now describe the three application-level adaptation procedures that we have developed to date.

A. The GARA Testbed

All experiments reported below were performed in the testbed shown in Figure 2. The testbed consists of three Cisco 7507 routers interconnected with 155 Mb/s (OC-3) ATM. Hosts are connected to the routers with 100 Mb/s switched Ethernet. All hosts used in our tests were Sun Ultra 60s. In addition, virtual circuits to several remote sites permit wide area experiments.

Cisco’s Modular QoS command line interface (MQC) is used for two different purposes. On the ingress interfaces to the network, it is used to classify, police, and mark packets. Within the interior of the network, it is used to enable Weighted Fair Queuing (WFQ) to give priority to marked packets.

B. Adaptive QoS Reservations: UDP Flows

We first describe how adaptive techniques can be used to determine the bandwidth reservation required to support a particular UDP flow. The motivation for this use of adaptation is that

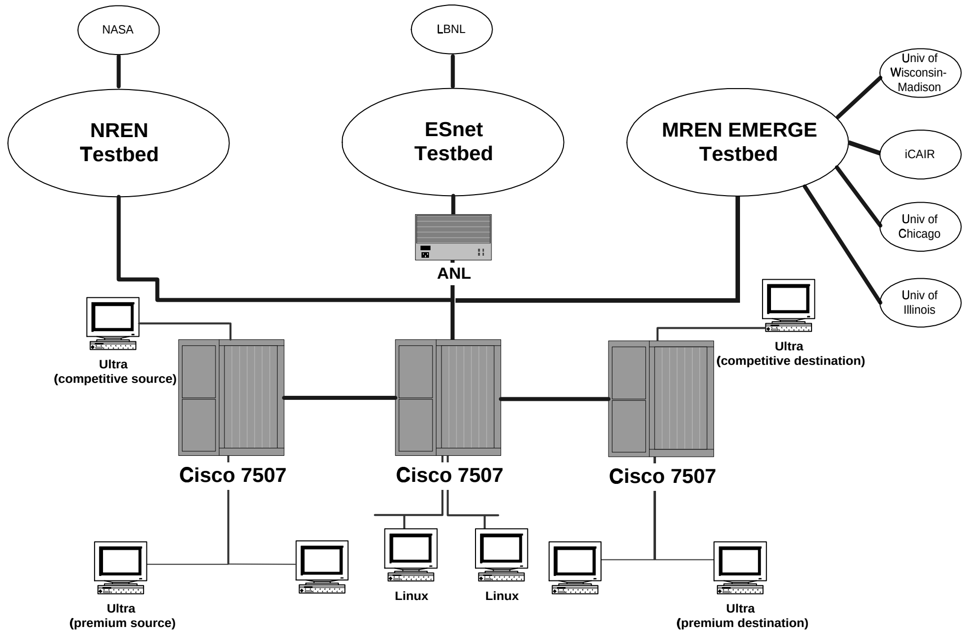

Fig. 2. The GARA Network Testbed (GARNET). The core of the testbed consists of three Cisco 7507 routers. There are several computers on each end of the testbed, more than shown here. Note the connections to three wide area networks.

many application developers have no knowledge or QoS mechanisms or of the principles by which QoS parameters are determined. We show that information provided by a simple packet loss rate sensor can be used to guide a decision procedure that sets bandwidth reservations adaptively, increasing reservations until loss rates reach zero. This decision procedure can be incorporated in an application or in a separate agent.

Our decision procedure uses information provided by the packet loss rate sensor described in Section III-C. Recall that this sensor periodically generates an estimate of the fraction of packets dropped, P; hence, $1 - P$ is the fraction of packets that conformed to the reservation. Our decision procedure calculates what reservation would have been needed to make such that no packets would have been dropped, as follows:

$$R_n(1-P) = R_o$$

or

$$R_n = \frac{R_o}{1-P}$$

where R_o is the old reservation and R_n is the new reservation.

To evaluate the effectiveness of this strategy, we performed experiments as follows. In order to obtain a replicable experiment, we used as our application a test program that sends UDP traffic at a user-specified rate across our testbed.

Results for two similar experiments are superimposed in Figure 3. In each case, the application made an initial reservation for 2500 kilobytes per second (KB/s) but then sent data at a higher rate: in the first case at 4000 KB/s and in the second case at 8000 KB/s. As described before, the first router classified, policed, and marked traffic. Because the router allows small bursts, the application initially was able to send slightly faster than the reservation allowed, but then the data rate settled down to a constant 2500 KB/s.

Our loss rate sensor is implemented by the GARA resource manager, which queries the router every ten seconds and provides feedback to the application for every query except the first. (The first query is not reported to the application because we wish to gather statistically sufficient data.) As the resource manager and application are not synchronized in any way, we should not be surprised that the feedback arrives at slightly different times in the two cases: at 16 seconds and 22 seconds, respectively.

It is clear from Figure 3 that the UDP application was able to adapt quickly in these experiments. However, the poor temporal resolution offered by our routers means that adaptation need not always work so well. For example, if the router statistics were gathered just as a series of packets were starting to be dropped, a unrepresentative result may be reported to the application. However, this problem would be compensated for after another round of adaptation. In addition, our router updates statistics only every ten seconds, which limits the frequency at which the resource manager can check them.

C. *Adaptive QoS Reservations: TCP Flows*

We should not be surprised that it is possible to determine UDP transmission rates by monitoring packet loss information, given that UDP does not perform congestion control. Implementing a comparable adaptive strategy for TCP is significantly more complex because of TCP's self-clocking mechanisms. Data that an application attempts to write into a socket buffer with a specific rate may not be transported immediately because TCP's sliding window protocol requires that acknowl-

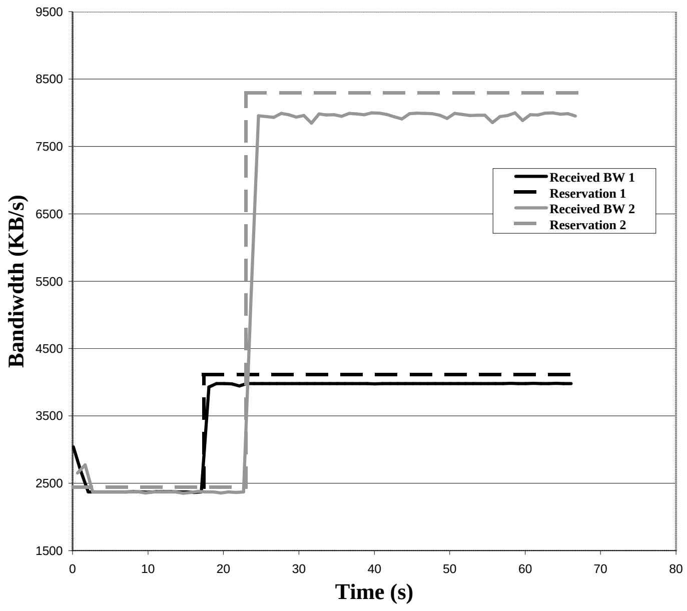

Fig. 3. Performance of our UDP reservation adaptation strategy in two different cases. In the first case, the application is sending at 4000 KB/s while in the second it is sending at 8000 KB/s. In both cases, an initial reservation of approximately 2500 KB/s is corrected after a single round of adaptation.

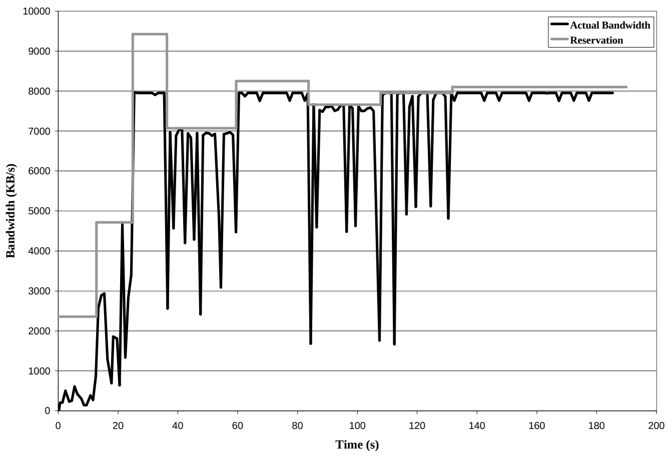

Fig. 4. An example of how our search-based reservation rate strategy determines the correct reservation to use for a TCP application. The application doubled its reservation request twice, then narrowed its reservation with a binary search. The difference between the gross and net reservations is due to packet overheads.

edgments be received before further data is sent. Also, TCP slows its sending rate when it believes it has encountered congestion. (In our case, TCP has not encountered congestion, but an aggressive QoS policing mechanism.) Nevertheless, TCP is used extensively in the applications that interest us, and so it is important to support TCP if we can.

Because of these difficulties, our decision procedure for TCP does not attempt to derive the transmission rate from the packet loss rate ratio. Instead, it uses a search procedure to determine the correct rate. When the packet loss rate sensor signals that packets have been dropped, we simply double the reservation. Once the reservation is large enough, we perform a binary search between the current reservation and the previous reservation until we arrive at a reservation that works and that has not changed from the previous reservation by more than five percent.

Figure 4 illustrates the results that we obtain with this heuristic. We see that the search takes some time to adapt but eventually comes close to a correct value. The time delay is largely because statistics on dropped packets are reported only every 10 seconds on our routers. Clearly, decreasing that interval would improve the adaptation time. Nevertheless, even this relatively long adaptation time is quite acceptable for many of our long-lived target applications.

There are a couple of possibilities for improving upon this search. One possibility is to change the initial doubling by estimating the correct multipler from the percentage of dropped packets, much as we did in the UDP case. We have performed extensive experiments with such techniques but have not yet succeeded in identifying a good estimate for the multiplier, because of TCP's complex behavior. Recent work proposes modifying TCP's windowing algorithm to be aware of reservation rates [17], [18].

It may be possible to adapt more quickly by monitoring closer to the application. In particular, if the application used an instrumented TCP library that could measure the rate at which the application was attempting to send, it could use the adjustment to adapt much more quickly. However, this strategy requires modifications to the application: our heuristics have the advantage of being usable by a third party agent.

D. A Bulk-Data Transfer Application

Our third example of an application-level adaptation procedure uses our BDT reservation-change sensor to guide rate adaptation for BDT applications. As described in Section III-D, this sensor signals changes in backgrond flow reservations due to preemption by (or termination of) higher-priority foreground flows. Our decision procedure simply adapts the transmission rate of the TCP-based bulk data transfer application in order to achieve throughput close to the bandwidth allocated to the BDT flow. Note that in the absence of this decision procedure, the achieved throughput would tend to be extremely low, because preemption lowers the background flow's reservation and packets that exceed the reservation are dropped, therefore triggering TCP's backoff algorithms.

For our experiments, this decision procedure was incorporated into a QoS-aware TCP-based BDT application. Figure 5 shows results obtained in a wide area testbed between Argonne National Labratory and Lawrence Berkeley National Labratory. At about time 5, the (background) BDT application began and was assigned all of the premium bandwidth, 25 MB/s. At approximately times 40 and 100, a foreground reservation began and the BDT reservation was reduced. When the foreground reservations ended, the background reservation was increased. Notice that at time 15, competitive UDP traffic began but does not interfere with either the foreground or background reservations.

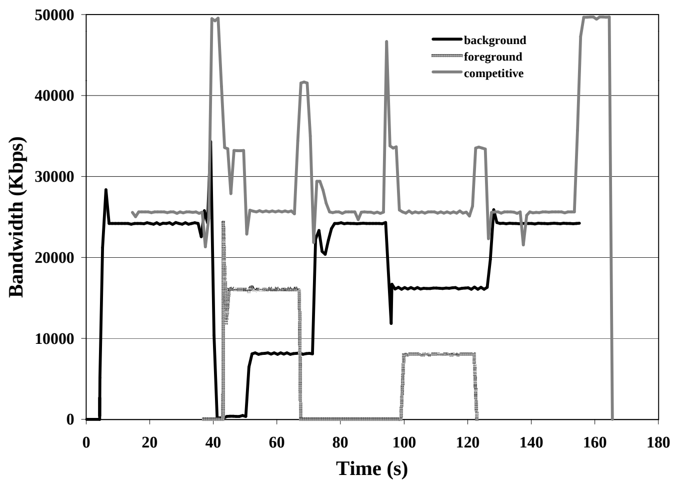

Fig. 5. An example of bulk-transfer in our wide area testbed. See text for details.

These results show that we are successful in adapting the BDT flow in response to information concerning preemption by foreground flows. Apart from a few artifacts, the BDT flow maintains data transfer at a rate close to the amount of premium bandwidth allocated to that flow. The artifacts can be explained as follows. First, we see that each time the BDT reservation is reduced, the BDT rate drops momentarily more than expected and then recovers. We attribute this behavior to the fact that TCP shrinks its window size when packets are dropped (when the reservation is changed before the application adapts), either by falling into its slow-start phase or into its congestion avoidance phase.

In addition, the application is using large socket buffers to obtain high performance over the wide area testbed and when it enters slow-start mode (because packets have been dropped once the reservation decreases) these socket buffers quickly fill up. As TCP increases its congestion window size exponentially during the slow-start phase, data is immediately available to send, and TCP sends the data as increasingly larger bursts, until the socket buffer is emptied. Because the former congestion window size did not reflect the actual amount of data transmitted, the length of the slow-start phase after a drop is too long, therefore, data is initially sent too rapidly for the updated router configuration, forcing packets to be dropped and TCP to go into slow-start mode again, until the congestion window becomes more appropriate. This effect is magnified by the larger bandwidth-delay product and hence larger socket buffers (1 MB in this case) in the wide area network.

V. Related Work

There has been a great deal of research on rate adaptation for network applications when reservation mechanisms are not present. For example, Goel et al. [7] describe a modular framework that provides feedback for not only network streams but also CPU scheduling. The present paper takes its terminology of actuators, sensors, and decision procedures from another feedback infrastructure, Autopilot [14], which has been used for dynamic performance tuning in various settings, including I/O. Our approach follows the concept of detaching the "controller" from the the application, as proposed in [3].

Implementing QoS-aware middleware is addressed in several projects. The Adaptive Quality of Service Architecture for distributed multimedia applications (AQUA) [19] introduces abstract interfaces for QoS measurements and negotiation. However, this work focuses on ATM-connections and how to ensure QoS under competition on the end-system.

The Quartz architecture [20] provides a CORBA-based QoS framework. It introduces agent-based adaption and a resource trader, called a balancing agent, which tries to compensate for the loss of resources by increasing the amount requested.

VI. Conclusions and Future Work

We have argued that advanced network applications such as teleimmersion, bulk data transfer, and distance visualization can benefit from mechanisms that enable the coordinated use of reservation and adaptation, via support for dynamic feedback among entities involved in making resource management decisions. We have described an implementation of such mechanisms within the GARA resource management architecture. In this implementation, sensors associated with resource and resource managers permit application-level monitoring of resource state and reservation status, while online control mechanisms enable adaptive control of reservations. We have used these mechanisms to develop three different application-level adaptive control mechanisms: two that use loss rate information to adapt reservations and one that uses reservation state information to adapt transmission rate.

We find these initial results encouraging, but recognize that much more work remains to be done. For example, we would like to experiment with more sophisticated resource-side allocation policies and determine to what extent applications can adapt to these policies in interesting ways. In more complex multidomain environments, performance feedback and adaptation become more complex, not least because relevant sensor information may not be easily accessible. Finally, experimentation with a wider range of applications is required.

Acknowledgments

We gratefully acknowledge assistance given by Linda Winkler and Becca Nitzen with the testbed used in these experiments and by Andy Adamson who wrote the UDP traffic generator. Numerous discussions with our colleagues Gary Hoo, Bill Johnston, Carl Kesselman, and Steven Tuecke have helped shape our approach to quality of service. We also thank Cisco Systems for an equipment donation that allowed the creation of the GARNET testbed. This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38; by the Defense Advanced Research Projects Agency under contract N66001-96-C-8523; by the National Science Foundation; and by the NASA Information Power Grid program.

References

[1] L. Wolf and R. Steinmetz, "Concepts for reservation in advance," *Kluwer Journal on Multimedia Tools and Applications*, vol. 4, May 1997.

[2] D. Ferrari, A. Gupta, and G. Ventre, "Distributed advance reservation of real-time connections," *ACM/Springer Verlag Journal on Multimedia Systems*, vol. 5, no. 3, 1997.

[3] B. Li and K.Nahrstedt, "A Control-based Middleware Framework for Quality of Service Adaptations," *IEEE Journal of Selected Areas in Communications, Special Issue on Service Enabling Platforms*, June 1999.

[4] B. Li and K. Nahrstedt, "QualProbes: Middleware QoS Profiling Services for Configuring Adaptive Applications," in *Proceedings of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware 2000)*, 2000.

[5] X. Wang and H. Schulzrinne, "Comparison of Adaptive Internet Multimedia Applications," *Institute of Electronics, Information and Communication Engineers Transactions*, vol. E82-B, pp. 806–818, June 1999.

[6] D. Sisalem and H. Schulzrinne, "The Loss-Delay Adjustment Algorithm: A TCP-friendly Adaptation Scheme," in *Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, July 1998.

[7] A. Goel, D. Steere, C. Pu, and J. Walpole, "Adaptive Resource Management Via Modular Feedback Control," Tech. Rep. 99-03, Oregon Graduate Institute, Computer Science and Engineering, Jan. 1999.

[8] W. Almesberger, J. L. Boudec, and T. Ferrari, "Scalable Resource Reservation for the Internet," in *IEEE Conference on Protocols for Multimedia Systems –Multimedia Networking*, Nov. 1997.

[9] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, "A Resource Allocation Model for QoS Management," in *18th IEEE Real-Time System Symposium*, 1997.

[10] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy, "A Distributed Resource Management Architecture That Supports Advance Reservations and Co-Allocation," in *International Workshop on Quality of Service*, pp. 27–36, June 1999.

[11] I. Foster, A. Roy, V. Sander, and L. Winkler, "End-to-End Quality of Service for High-End Applications," tech. rep., Argonne National Laboratory, 1999. `http://www.mcs.anl.gov/qos/qos_papers.htm`.

[12] I. Foster and C. Kesselman, eds., *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, 1999.

[13] I. Foster, J. Insley, G. von Laszewski, C. Kesselman, and M. Thiebaux, "Distance Visualization: Data Exploration on the Grid," *IEEE Computer Magazine*, pp. 36–43, Dec. 1999.

[14] R. L. Ribler, J. S. Vetter, H. Simitci, and D. A. Reed, "Autopilot: Adaptive Control of Distributed Applications," in *Proc. 7th IEEE Symp. on High Performance Distributed Computing*, IEEE Computer Society Press, 1998.

[15] K. Nichols, V. Jacobson, and L. Zhang, "A Two-Bit Differentiated Services Architecture for the Internet," *Internet RFC 2638*, July 1999.

[16] S. Blake, D. Black, M. Carlson, M. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," *Internet RFC 2475*, 1998.

[17] W. Feng, D. Kandlur, D. Saha, and K. Shin, "Understanding and Improving TCP Performance Over Networks with Minimum Rate Guarantees," *IEEE/ACM Transactions on Networking*, vol. 7, pp. 173–187, Apr. 1999.

[18] I. Yeom and A. N. Reddy, "Realizing Throughput Guarantees in a Differentiated Services Network," in *IEEE Int. Conf. on Multimedia Computing and Systems*, pp. 372–376, June 1999.

[19] K. Lakshman and R. Yavatkar, "Integrated CPU and Network I/O QoS Management in an End-System," *Intel Architecture Labs and University of Kentucky in Computer Communications Journal, Special Issue on Quality of Service in Distributed Systems*, vol. 21, Apr. 1997.

[20] F. Siqueira and V. Cahill, "Delivering QoS in Open Distributed Systems," in *Proceedings of the 7th IEEE Workshop on Future Trends in Distributed Computing Systems (FTDCS'99)*, Dec. 1999.