End-to-End Quality of Service for High-End Applications

Ian Foster[ab], Markus Fidler[c], Alain Roy[d], Volker Sander[e], Linda Winkler[a]

[a]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, U.S.A.

[b]Department of Computer Science, The University of Chicago, Chicago, IL 60637, U.S.A.

[c]Department of Computer Science, Aachen University, 52064 Aachen, Germany

[d]Department of Computer Sciences, University of Wisconsin-Madison, Madison, WI 53706, U.S.A.

[e]Central Institute for Applied Mathematics, Forschungszentrum Jülich GmbH, 52425 Jülich, Germany

High-end networked applications such as distance visualization, distributed data analysis, and advanced collaborative environments have demanding quality of service (QoS) requirements. Particular challenges include concurrent flows with different QoS specifications, high bandwidth flows, application-level monitoring and control, and end-to-end QoS across networks and other devices. We describe a QoS architecture and implementation that together help to address these challenges. The General-purpose Architecture for Reservation and Allocation (GARA) supports flow-specific QoS specification, immediate and advance reservation, and online monitoring and control of both individual resources and heterogeneous resource ensembles. Mechanisms provided by the Globus Toolkit are used to address resource discovery and security issues when resources span multiple administrative domains. Our prototype GARA implementation builds on differentiated services mechanisms to enable the coordinated management of two distinct flow types—foreground media flows and background bulk transfers—as well as the co-reservation of networks, CPUs, and storage systems. We present results obtained on a wide area differentiated services testbed that demonstrate our ability to deliver QoS for realistic flows.

1. Introduction

Investigations of network quality of service (QoS) have tended to focus on the aggregation of relatively low-bandwidth flows associated with Web and media streaming applications. Yet the QoS requirements associated with these flows are not representative of all interesting applications. For example, distance visualization applications encountered in science and engineering can involve data transfers and media streaming at hundreds (ultimately thousands) of megabits per second (Mb/s), while the bulk data transfer operations required for replication or analysis of large datasets can require sustained high bandwidths expressed in terms of terabytes per hour. Advanced collaborative environments can require complex mixes of these and other flows, with varying service level requirements and many interdependencies.

The development of QoS for such high-end applications introduces major challenges for both QoS protocols and higher-level architectures that use these protocols to provide end-to-end solutions for users.

At the higher-level architecture level, new concepts and constructs are required for dealing with end-to-end flows that involve multiple scarce resources: for example, advance reservation mechanisms, to ensure availability [14,17, 52]; co-reservation of network, compute, storage, and other resources [11]; control and monitoring application programmer interfaces (APIs) for application-level adaptation [26,15,45]; and policy mechanisms able to deal with large reservations and complex hierarchical allocation strategies.

When considering appropriate QoS protocols to support these high-end applications, two major contenders are apparent: Integrated Services

and Differentiated Services. The Integrated Services architecture [7] aims at addressing these heterogenous demands for quality of service by allowing end-to-end reservations of network capacity for individual flows, usually by using the Resource Reservation Protocol (RSVP) [53]. Unfortunately, the fine granularity of the Integrated Services approach as originally specified was unlikely to scale effectively to be widely used in the Internet. (Note however recent proposals that avoid the need to police all flows [35].) In addition to attempted modifications of Integrated Services, Differentiated Services (DS) [6], which is the most recent approach of the Internet Engineering Task Force (IETF) towards quality of service, addresses these scalability issues by an aggregation of micro-flows to classes. Doing so allows to support only a small number of service classes within the core network and thereby offers better scalability. While DS has advantages in terms of scalability, it is not obvious whether and how it can support specialized high-end flows.

The work that we present in this article addresses both the higher-level architecture and protocol challenges just described. We describe the General-purpose Architecture for Reservation and Allocation (GARA), a resource management architecture that builds on mechanisms provided by the Globus Toolkit [19] to support secure immediate and advance co-reservation, online monitoring/control, and policy-driven management of a variety of resource types, including networks [21]. Then, we describe the application of GARA concepts and constructs to DS networks. We present a DS resource manager (i.e., bandwidth broker [6,31]) and explain how this resource manager integrates with GARA facilities (e.g., advance reservation, authentication/authorization). We describe how this resource manager builds on DS mechanisms to support two heterogeneous types of flows within a single framework: latency- and jitter-sensitive (e.g., media flows) and high-bandwidth but latency-insensitive (e.g., bulk transfer). We also propose a policy model that allows admission control decisions to be made at multiple levels. Finally, we present performance experiments conducted on both local area and wide area DS network testbeds; our results demonstrate our ability to support multiple flow types and to co-reserve network and CPU resources.

The rest of this article is structured as follows. In Section 2 we introduce the QoS requirements that high-end applications have. Then, in Section 3, we describe GARA and its implementation. In Section 4, we discuss its application in the context of DS networks and in Section 5 we present our experimental results. We discuss multi-domain issues in Section 6 and related work in Section 7, and conclude with a discussion of future directions.

2. QoS Requirements of High-End Applications

We use three representative examples to illustrate QoS requirements of the high-end network applications that are encountered, for example, in advanced scientific and engineering computing [20].

2.1. Application Descriptions

Distance visualization of large datasets. Scientific instruments and supercomputer simulations generate large amounts of data: tens of terabytes today, petabytes within a few years. Remote interactive exploration of such datasets requires that the conventional visualization pipeline be decomposed across multiple resources [1,5,18]. A realistic configuration might involve moving data at hundreds or thousands of Mb/s to a data analysis and rendering engine which then generates and streams real-time MPEG-2 (or later HDTV) video to remote client(s), with control information flowing in the other direction. QoS parameters of particular interest for this class of application include bandwidth, latency, and jitter; resources involved in delivering this QoS include storage, network, CPU, and visualization engines.

Large data transfers. In other settings, large datasets are not visualized remotely but instead are transferred in part or in their entirety to remote sites for storage and/or analysis [33,4,8,32]. The need to coordinate the use of other resources with the completion of these multi-gigabyte or terabyte transfers leads to a need for QoS guar-

antees of the form "data delivered by deadline" rather than instantaneous bandwidth. Achieving this goal requires the scheduling of storage systems and CPUs as well as networks so as to achieve often extremely high transfer rates.

High-end collaborative environments. High-end collaborative work environments involve immersive virtual reality systems, high-resolution displays, connections among many sites, and multiple interaction modalities including audio, video, floor control, tracking, and data exchange. For example, the Argonne "Access Grid" currently connects some 15 sites via multiple audio, video, and control streams, with the audio streams especially vulnerable to loss. Such applications require QoS mechanisms that allow the distinct characteristics of these different flows to be represented and managed [13,24].

2.2. QoS Requirements

Heterogeneous flows. The applications of interest frequently incorporate multiple flows with widely varying characteristics, in terms of bandwidth, latency, jitter, reliability, and other requirements. GARA addresses these requirements through (a) support for per-flow QoS specifications while maintaining DS-like scalability and (b) a QoS-mechanism-independent architecture that adapts to multiple techniques. A common API means that for example a distance visualization application can specify the distinct requirements of high-volume data and latency-sensitive control flows, in a mechanism-independent manner; these flows might then be mapped to different mechanisms: e.g., Multi-Protocol Label Switching (MPLS) [41] and DS.

High bandwidth flows. Some applications involve high bandwidth flows that may require a large percentage of the available bandwidth on a high-speed link. For example, we and others have demonstrated transfer rates of over a Gb/s over wide area networks. This characteristic has significant implications for both mechanisms and policy. QoS mechanisms are required that can support such flows while allowing coexistence with other flows having different characteristics. At the policy level, we believe that approaches are required that allow for the coordinated management of resources in multiple domains, so that virtual organizations (e.g., a scientific collaboration) can express policies that coordinate the allocation of the resources available to them in different domains.

Need for end-to-end QoS. Satisfying application-level QoS requirements often requires the coordinated management of resources other than networks: for example, a high-speed data transfer can require the scheduling of storage system, network, and CPU resources. As we shall see, GARA addresses this requirement by defining an extensible architecture that can deal with a range of different resource types and by providing support for the co-allocation of multiple resources.

Need for application-level control. High end-to-end performance requires that applications be able to discover resource availability (GARA), monitor achieved service, and modify QoS requests (to network and to other resources, such as CPUs–for example, to reduce reservations when load drops [16]) and application behavior dynamically.

Need for advance reservation. Specialized resources required by high-end applications such as high-bandwidth virtual channels, scientific instruments, and supercomputers are scarce and in high demand; in the absence of advance reservation mechanisms, coordination of the necessary resources is difficult. Reservation mechanisms are needed to ensure that resources and services may be scheduled in advance. Snell et al. have shown that a meta-scheduler, which schedules a set of Grid resources, can improve the overall effectiveness of the Grid by requesting a deterministic resource in advance [44].

3. The GARA QoS Architecture

We designed GARA to meet the QoS requirements listed above. We introduce GARA concepts here, and then describe below how we apply these concepts in DS networks to manage the allocation of a particular flavor of QoS capability, namely Premium and Guaranteed Rate service.

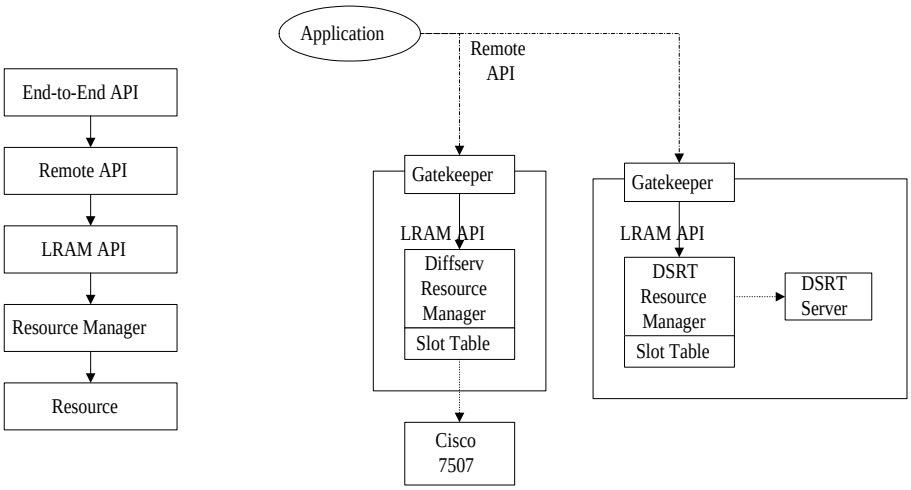

Figure 1. On the left, the principal APIs used within GARA. On the right, the principal components of the GARA prototype as instantiated for DS and DSRT (CPU scheduler) services, with our own resource manager and slot manager being used in both cases. In the DS case, commands are issued to a router while in the DSRT case commands are issued to a DSRT server (for tracking reservations).

3.1. GARA Overview

GARA defines APIs that allow users and applications to manipulate reservations of different resources in uniform ways. For example, essentially the same calls are used to make an immediate or advance reservation of a network or CPU resource. Once a reservation is made, an opaque object called a *reservation handle* is returned that allows the calling program to monitor, modify, and cancel the reservation. Other functions allow reservations to be monitored by polling or through a callback mechanism in which a user's function is called every time the state of the reservation changes in an interesting way.

As illustrated on the left side of Figure 1, GARA defines APIs at multiple levels so as to maximize both the functionality delivered to the user and opportunities for code reuse in implementations. In particular, the Local Reservation and Allocation Manager (LRAM) API provides direct access to reservation functions within a trust domain, while the remote API provides remote access to LRAM functionality, addressing issues of authentication and authorization. Both APIs implement the functionality described in the preceding paragraph.

The uniform treatment of reservations provided by GARA makes it possible to define and reuse co-reservation and co-allocation libraries that encode strategies for the coordinated use of multiple resources [11]. Because different resources (e.g., computers and storage systems) can be manipulated via the same function calls, standard libraries can be developed that encode strategies for dealing with, for example, co-reservation and fault recovery.

One co-reservation library that we have developed in support of our work with GARA implements an end-to-end network API that provides end-to-end analogs of each of the remote API calls. This API allows the user to create, monitor, cancel, etc., network co-reservations: that is, reservations involving more than one network resource. This API allows users and applications to ignore details of the underlying network topology.

Figure 2 illustrates the use of this end-to-end

API. This program first determines the bandwidth requirements of an application and then queries to determine available Premium bandwidth over the path of interest. A reservation is created for the smaller of these two values and the reservation handle `H` is used to bind the reservation to the previously created flow. The application then checks periodically to see whether the reservation can be increased. Notice that the changes to what is otherwise a conventional socket-based code are small.

```
UDP-streamer(host A, host B) {
  (PortA,PortB) = new_socket_conn(A,B)
  F = compute_flow_requirement()
  When = {NOW,60 mins}
  Max = EnquireE2EResv(A,B,When)
  if (Max.forward > F) then
     R = F
  else
     R = Max.forward
  H = CreateE2EResv(A,B,R,0,When)
  BindE2EResv(H, PortA, PortB)
  repeat until done {
     <send for a while>
     Max = EnquireE2EResv(A,B,When)
     if (Max.forward > 0 && R < F) then {
        R = Max.forward + R
        if (R > F) then
           R = F
        ModifyE2EResv(H, R, When)
     }
  }
}
```

Figure 2. Pseudo-code for a simple application that uses the GARA end-to-end API to first make and subsequently monitor and modify a reservation. For brevity, this code does not include error checking.

We note that while this example emphasizes application-centered monitoring and control of reservation state, GARA also supports third-party reservation operations. For example, we could remove the reservation logic from Figure 2 altogether and instead perform appropriate reservation operations in a separate process.

3.2. GARA Implementation

We review GARA implementation issues and status, working up from the bottom of our API stack.

GARA must provide admission control and reservation enforcement for multiple resources of different types. Because few resources provide reservation capabilities, we have implemented our own resource manager so as to ensure availability of reservation functions. As illustrated in Figure 1, this manager uses a slot table [14,31] to keep track of reservations and invokes resource-specific operations to enforce reservations. Requests to this resource manager are made via the LRAM API and result in calls to functions that add, modify, or delete slot table entries; timer-based callbacks generate call-outs to resource-specific routines to enable and cancel reservations. Note that only certain elements of this resource manager need to be replaced to instantiate a new resource interface. To date, we have developed resource managers for DS networks (described below), for the Distributed Soft Real-Time (DSRT) CPU scheduler [9], and for the Distributed Parallel Storage System (DPSS) [50], a network storage system; others are under development.

Our implementation of the end-to-end API invokes a path service to identify the resource managers that must be contacted to arrange for an end-to-end reservation, and then makes a series of GARA remote API calls to perform the co-reservation operation. See below for a discussion of issues that arise when traversing multiple domains.

Our GARA prototype uses two "Grid" services provided by the Globus Toolkit: the Monitoring and Discovery Service (MDS) [10], currently based on the Lightweight Directory Access Protocol (LDAP), which is used for publishing reservation status information and for accessing path information; and the public-key based Grid Secu-

rity Infrastructure for authentication and authorization services. The interfaces to these services are simple and well-defined (LDAP and GSS-API, respectively), hence it is straightforward to substitute alternative implementations.

4. GARA and Differentiated Services Networks

The DS architecture is based on a simple model in which packets entering a network are classified and possibly conditioned at the boundaries of the network by service provisioning policies, and assigned to different behavior aggregates. Within the core of the network, packets are forwarded according to the per-hop behavior (PHB) associated with the DS classification. These mechanisms have the advantage of not requiring that per-flow state be maintained within the network. However, few guarantees can be made about end-to-end behaviors, which instead emerge as the composition of the PHBs associated with individual links.

4.1. Integrating Differentiated Services and GARA

We have interfaced GARA concepts and constructs to DS mechanisms in order to manage the allocation of Premium or Guaranteed Rate service bandwidth. As shown in Figure 4, we associate GARA resource managers with the locations at network edges where admission control occurs. These resource managers are, in essence, what DS papers call "bandwidth brokers" [6]: they generate their region's marked (Premium) traffic allocations and control the devices (e.g., routers) used to enforce these allocations. Requests to resource managers are authenticated, ensuring secure operation.

We have constructed our DS resource manager to support two classes of Premium service: a foreground service, for latency- and jitter-sensitive flows (e.g., multimedia streaming and control), and a background service, for long-lived, high bandwidth but latency-insensitive flows (e.g., bulk data transfer operations). The resource manager changes background reservations dynamically as foreground reservations come and go, generating callbacks to the application when a reservation changes. This strategy allows bulk data transfers to co-exist with multimedia flows. The amount of bandwidth available for background reservations over a particular time period can then be controlled via policy mechanisms. We report results with this approach below. Our prototype supports multiple foreground reservations but initially only a single background reservation; the extensions required to support multiple background flows are not complex.

A resource management framework for DS networks must also address end-to-end issues. A typical wide area flow requires allocations of Premium bandwidth at multiple edge routers and also within interior domains. For example, in Figure 4, a Premium flow from ANL to LBNL should, in principle, require an allocation not only from the ANL domain for the ANL/ESnet interface (where marking occurs) but also from ESnet for the ANL-LBNL transit traffic and from the LBNL domain for the ESnet/LBNL interface. Hence, we need to associate resource managers with multiple DS domains and to implement co-reservation strategies. Co-reservation operations must be designed with end-to-end verification in mind. In our example, an application that omitted to obtain a reservation for ESnet transit traffic could cause problems for other ANL-LBNL traffic, for example if the aggregate ANL-ESnet traffic exceeded what was allowed by the current ANL-ESnet service level agreement (SLA).

Most DS work assumes that co-reservation operations are encapsulated in the local domain's resource manager: hence, a request to reserve bandwidth from ANL to LBNL results in the ANL manager contacting the ESnet manager, which in turn contacts the LBNL manager. Upon receipt of a positive response from both other managers, a reservation is established. This approach has the advantages of providing trusted co-reservation and of encapsulating all bandwidth broker communication within a single local entity. The approach has disadvantages in settings where end-to-end reservations involve resources other than networks, as a hierarchical co-reservation structure results, or where allocation policies at interior domains depend on factors other than the

identity of the requesting manager.

An alternative approach to this problem is to define a two-phase commit protocol. In this approach, an application program—or agent working on behalf of an application program—contacts each manager in turn. In the first phase, a manager can indicate that acceptance of a reservation is conditional on the requestor securing acceptance (indicated by a signed certificate) from the next manager.

In both approaches, inter-domain SLAs can either be established statically (in which case reservations can only be made if they fit within the pre-established SLAs), or they can be established dynamically, as reservations are made. The latter approach provides greater flexibility but requires more sophisticated policy and enforcement engines in interior domains, as discussed below.

Our initial GARA prototype implements neither of the approaches just described but instead relies on the end-to-end library to implement coreservations correctly. We assume two domains and static SLAs between domains; hence, we need to allocate bandwidth at just two locations. Reservation policies are expressed via access control lists associated with individual resource managers. These limitations are not inherent in our model and are being removed in current work.

4.2. Differentiated Service Configuration

The final issue to be addressed in a DS implementation relates to how PHBs are configured to provide the premium services desired for particular applications. In our DS implementation, this set-up involves the use of Committed Access Rate (CAR) and Weighted Fair Queuing (WFQ) mechanisms configured by means of the Modular Quality of Service Command-Line interface (MQC) [51]. Figure 3 shows a schematic of the functionality that is applied at ingress routers.

We police the guaranteed rate traffic on the ingress ports of edge routers and mark all conforming packets by setting the DS Code-Points defined in [39]. An over-provisioned Weighted Fair Queuing configuration is used on the egress port of all routers.

The operation of CAR is controlled via commands issued to the router by the associated GARA resource manager as reservations become active, terminate, are modified, or are cancelled. These commands enter, remove, or modify flow specifications that define a Premium service flow in terms of its source and destination IP address and port, and its rate limit specification (desired average transmission rate bandwidth and a normal and excess burst size). Communication from the resource manager to the Cisco Systems router is performed via Command Line Interface.

We also use CAR on the ingress ports of interdomain routers, where it is used to enforce SLAs negotiated with other domains, by rate limiting the marked premium traffic that will be accepted from another domain.

WFQ is used on the egress port of edge routers and in interior routers. WFQ ensures that in periods of congestion—i.e., when packets get queued in the router because the output link does not provide the capacity for delivering them immediately—each DS class receives at least the fraction of the output bandwidth given by the weight defined for that class. Hence, as long as the total marked traffic destined for an output port does not exceed the allocated output bandwidth, WFQ can be used to ensure that marked traffic is forwarded without delay despite congestion in other classes.

This use of CAR and WFQ approximates a Guaranteed Rate service which can be build either on top of the Expedited Forwarding (EF) PHB described by the IETF's DS Working Group in [12], or by using one of the Olympic services build on top of the Assured Forwarding (AF) PHB described in [29].

Applying CAR and WFQ raises the question of how these mechanisms should be configured to meet application-level QoS requirements. This question is complicated by the wide variety of flows that we wish to support (UDP, TCP, low and high bandwidth) and the geographic scale over which QoS is required: from a few meters to thousands of kilometers. Considerable experimentation on the testbed described in the next section has been performed to understand these issues.

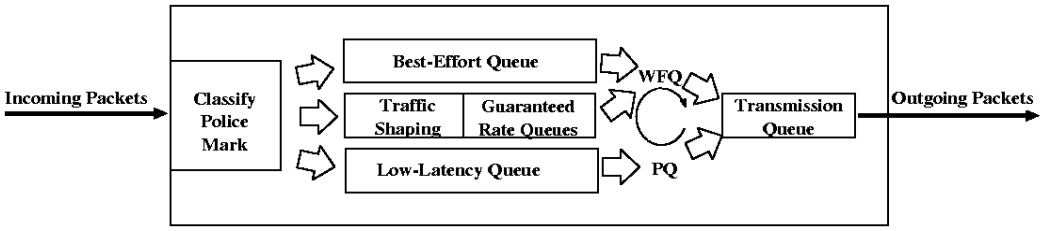

Figure 3. The ingress router configuration consists of a classification, policing and marking unit and of an optional traffic shaper. Traffic shaping is not applied for services that require a low latency, whereas guaranteed rate streams are shaped by applying a number of per-flow holding queues prior to WFQ scheduling.

5. Experimental Studies

We report on experiments designed to evaluate the effectiveness of both the GARA architecture and our DS implementation. In particular we show how a long lived bulk data transfer with deadline can share a Premium class with short-lived prioritized foreground flows that require explicit reservation. Two efficient implementations that support such applications are shown. One uses backward signaling, in which case the reservation manager provides information of currently unused Premium capacity to background bulk data transfer applications. These in return are instrumented to adapt their sending rate dynamically to the available Premium capacity. The other option allows the background bulk data transfer to make use of the resource manager's ability to pace the bulk data transfer traffic by the use of traffic shaping at the output interface of the ingress router. Hence, the transmission rate will be automatically updated by TCP's self clocking mechanisms.

5.1. Experimental Configuration

Our experimental configuration, illustrated in Figure 4, comprises a laboratory testbed at Argonne National Laboratory (the Globus Advance Reservation Network Testbed: GARNET) connected to a number of remote sites, including Lawrence Berkeley National Laboratory (LBNL). Connectivity to LBNL is provided by the Energy Sciences Network (ESnet) DS testbed. GARNET allows controlled experimentation with basic DS mechanisms; the wide area extensions allow for more realistic operation, albeit with a small number of sites. As end-system resources are located in different domains, we must deal with distributed authentication and authorization.

Cisco Systems 7507 routers are used for all experiments. Within GARNET, these routers are connected by OC3 ATM connections; across wide area links, they are connected by VCs of varying capacity. We are restricted to these relatively slow speeds because the 7507 cards do not implement CAR and WFQ at speeds faster than OC3. End system computers are connected to routers by either switched Fast Ethernet or OC3 connections. CAR and WFQ are used for QoS enforcement, as described above. Flow specifications supplied to CAR use a bandwidth computed from the user-specified required bandwidth, taking into account packet headers (note that this requires packet size information), with nonconforming traffic dropped. Burst size and excess burst size parameters are both set as follows: if using TCP, to the bandwidth (in bytes/second) times the assumed maximum round trip time, subject to a minimum of 8 Kbytes and a maximum of 2 Mbytes; if using UDP, to 1/4 of this value, subject to a minimum of 8 Kbytes. WFQ was configured statically in all experiments.

During the initial experiment, no traffic shap-

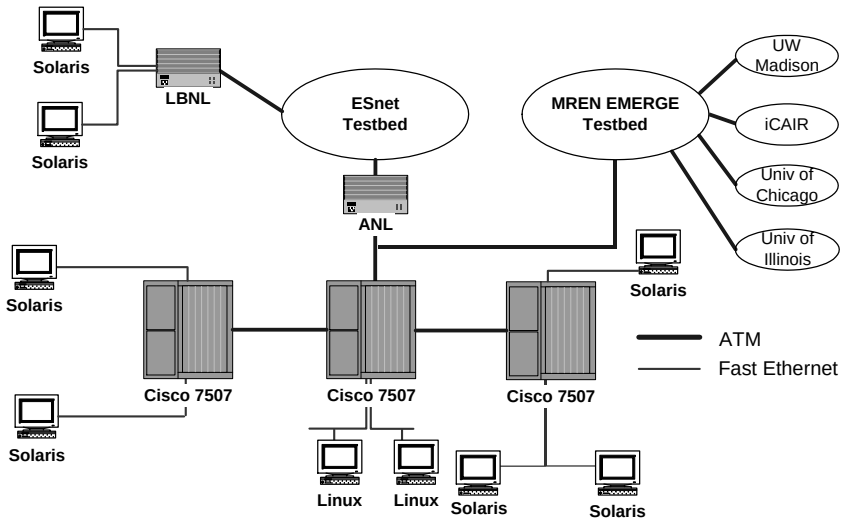

Figure 4. The experimental configuration used in this work, showing our local GARNET testbed and its extensions to remote sites connected via ESnet and MREN.

ing is performed on Premium flows beyond the limited shaping provided by WFQ in the presence of congestion. While the lack of shaping has not proven to be a significant problem to date, it will likely be required in future, more dynamic environments.

The network speeds supported in this testbed are clearly not adequate for the high-end applications discussed above: the largest Premium flow that we can support is around 80 Mb/s. Nevertheless, this testbed configuration has allowed us to validate multiple aspects of our general approach. We plan to extend our approach to higher-speed networks in future work.

5.2. Multiple Flows: Local Area Case

Our first experiments evaluate GARA's ability to support multiple flows simultaneously and to support application monitoring of, and adaptation to, changes in reservation status.

We first report on experiments conducted on our local GARNET testbed: see Figure 4. We configured GARNET to create a 45 Mb/s Premium channel in a 100 Mb/s network. We then created five distinct flows: a bulk data transfer, operating as a "background" flow; a competing 80 Mb/s Best-Effort UDP flow (a traffic generator submitting 1,000 byte packets every 100 μsecs); and three independent, short-lived foreground flows with immediate reservations. In this and subsequent experiments we used a simple data transfer program, a modified version of ttcp that was able to limit the frequency that it wrote to the socket buffer, in order to achieve a user-specified bandwidth, as our "application." The source and destination computers used for the Premium flows were distinct from the computers used for the competing flows.

Figure 5 shows the bandwidth delivered to the foreground, background, and Best-Effort flows during the experiment. We succeed in delivering "excess" Premium bandwidth to the bulk transfer application without compromising the foreground flows. The good bulk transfer performance that was achieved was made possible by the resource manager's callbacks to the bulk transfer application, which allowed the application to change its sending rate in response to changes in its allo-

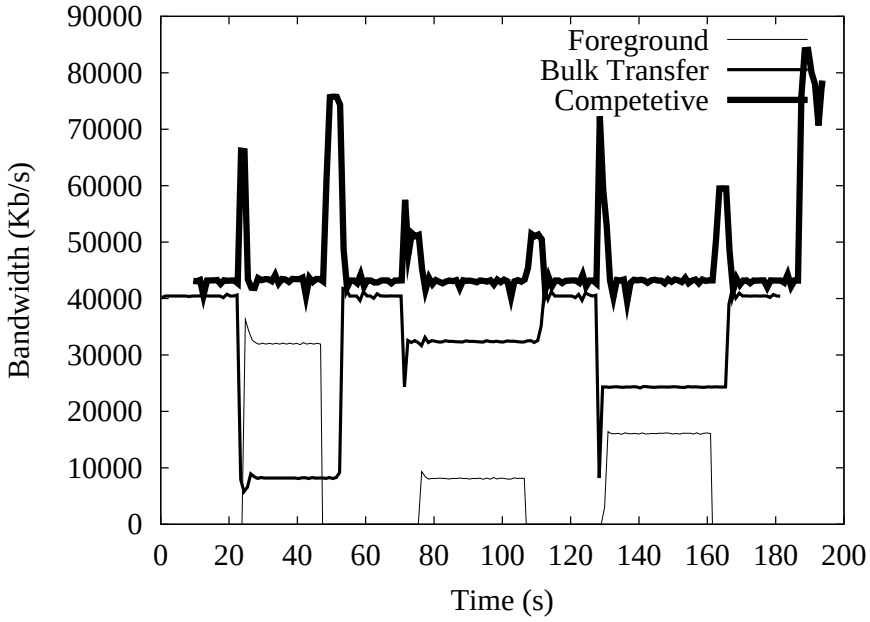

Figure 5. Performance achieved for a mixture of Premium and Best-Effort services on GARNET. We demonstrate that a bulk-transfer (background) application is able to exploit unused Premium traffic without affecting foreground reservations. See text for details.

cated bandwidth, thereby avoiding packet drops and the invocation of TCP slow-start. The following is a more detailed explanation of the graph:

1. The graph begins with the background TCP traffic, which has a bulk-transfer reservation. This flow is initially allocated 40.5 Mb/s Premium bandwidth: that is, 90 percent of the 45 Mb/s Premium traffic.

2. The competitive UDP traffic is started shortly after the bulk transfer but does not affect it due to the Premium status of the bulk transfer flow.

3. At 25 secs, another application makes an immediate 36 Mb/s reservation and initiates a 32 Mb/s foreground flow. A callback notifies the bulk transfer application, which reduces its sending rate to adapt to the reduced reservation. (The and other similar transitions take a little time due to the time required to control the router.)

4. At 48 secs, the foreground application finishes its transmission and then cancels its reservation. Another callback allows the bulk transfer process to increase its sending rate to adapt to the newly available Premium traffic.

5. Subsequently, two other foreground flows are created, with similar effects: a 9 Mb/s reservation (8 Mb/s flow) from 75 to 105 secs and an 18 Mb/s reservation (16 Mb/s flow) from 130 to 160 secs.

6. At time 185, the background flow completes and cancels its reservation. The competing traffic rate increases to its target of 80 Mb/s, actually exceeding this briefly because of the filled router queues.

Notice that each time the bulk transfer reservation is reduced, the bulk-transfer rate drops momentarily then recovers. We attribute this behavior to the fact that TCP shrinks its window size when packets are dropped, due to slow start or congestion avoidance.

The model of bulk transfers used to date is intended to increase the economic use of the Guaranteed Rate service. An important improvement to this mechanism would be to add a minimum reservation for the background traffic. By thus guaranteeing a particular amount of foreground traffic to the background class, deadline staging operations can easily be implemented. Again, the economic use indicates that the required reservation to meet the deadline might change over time. Whenever a status update of the available bandwidth is received by the resource manager, it can easily calculate the new required amount of bandwidth.

While figure 5 lists only a single bulk transfer flow, the basic mechanism can easily be extended to support multiple flows. Assume that there are n bulk transfer flows, with minimum bandwidths $b_0, \ldots, b_{n-1}$. At any given time, there is unused bandwidth U, and we know that

$$U \geq \sum_{i=0}^{n-1} b_i$$

This is because we do admission control to ensure that each bulk transfer flow will always have its minimum bandwidth. We will assign each actual bandwidth, B_i as:

$$B_i = U \cdot \frac{b_i}{\sum_{j=0}^{n-1} b_j}$$

This gives a proportional share of the bandwidth, and ensures that each flow gets its minimum bandwidth. An optimization could be to decay the minimum bandwidth when the application is able to send at rates greater than the minimum bandwidth, thus allowing more bandwidth to be given to other flows. This allows the links to be shared, while still assisting the bulk transfer flows in finishing as soon as possible.

We now extend this model of bulk data transfer by also applying for unused Best-Effort bandwidth. The required reservation of Premium capacity to meet a deadline can in particular be decreased during the run time of a bulk data transfer, if parallel sockets are used for the transmission. Thereby one of the sockets has to be configured to generate Premium traffic at the confirmed rate to ensure that the deadline is met, whereas no reservation of network capacity is made for the remaining sockets, but these are mapped to a less than Best-Effort service in order to allow for fairness among competing and responsive Best-Effort flows. Within the QBone Internet2 project the DS Scavenger service [47] was proposed recently to provide such a less than Best-Effort service for high-bandwidth flows. By applying the Scavenger service the bulk transfer application not only collects unused Premium bandwidth but also unused Best-Effort capacity at the prize, that the Scavenger service can be starved by Best-Effort traffic. Whenever the bulk transfer application succeeds in transmitting an appreciable amount of additional data by applying the Scavenger service, forward signaling of a reduced required Premium rate to the reservation manager can be applied to decrease the reserved Premium capacity and thus reduce costs and allow for a smaller blocking rate of the reservation manager, due to a higher available Premium capacity.

Our prototype implementation of such a parallel bulk data transfer application is based on a simple data fragmentation and the use of two parallel sockets with the proposed mapping of one socket to the Premium service and one to the Scavenger service. In order to overcome the effects of TCP congestion control in the Scavenger class, the implementation option of using multiple striped sockets for this class exists as this is offered by gridftp [3].

Figure 6 and 7 show results obtained from our testbed implementation of this combined use of a Premium and a Scavenger service. A target deadline for a 280 MByte file transfer of 200 seconds is applied. The required Premium rate to ensure this deadline with a safety margin of 10 seconds is derived to 12 Mb/s, for which an initial reservation is made. The sender in addition to this Guaranteed Rate service applies the Scavenger service in parallel, to utilize unused Premium and Best-

Effort capacity, if any is available. Each time the sender successfully transmits a configurable additional amount of data (25 MByte) by applying the Scavenger service, it recomputes the reservation of Premium capacity that is required to meet the deadline, and performs a reservation update.

In figure 6 a scenario without additional traffic across an ATM bottleneck link of roughly 42 Mb/s net capacity is addressed. Besides the guaranteed rate of 12 Mb/s used by the premium stream, the Scavenger stream can use the remaining capacity with a rate of about 30 Mb/s. After 7 seconds, the sender performs the first Premium service rate adaptation, due to the additional 25 MByte of data transferred by means of the Scavenger service, and lowers the Premium capacity reservation to about 10.5 Mb/s. Since the ATM bottleneck link in this experiment is not used by any other flow, the rate at which the Scavenger service can be used increases proportionately. This process occurs repeatedly during the file transfer and allows the reservation manager to redistribute the freed Premium capacity. In addition the use of parallel sockets reduces the file transfer time to 58 seconds.

Figure 7 shows the same scenario under congestion. After 10, 40 and 70 seconds, congestion occurs, each time for 10 seconds. During these periods the congestion leads to the intended starvation of the less than Best-Effort Scavenger service and thus achieves the desired fairness among Scavenger and Best-Effort flows. Nevertheless the deadline of the file transfer is never endangered, since the Premium flow still receives the currently required guaranteed rate. Due to the reduced amount of data transmitted by applying the Scavenger service, Premium rate adaptations can in this scenario be made less frequently, and an overall file transfer time of 88 seconds is measured, which still is well below the Premium flow target deadline of 190 seconds.

Therefore the use of parallel streams in a file transfer scenario and the mapping of these streams on a Guaranteed Rate and a Scavenger service achieves three main goals:

- The file transfer deadline is guaranteed.
- Excess Premium and Best-Effort capacity can be used if available, to reduce the overall transmission time, and to be able to free Premium resources earlier.
- Fairness towards the Best-Effort class is achieved by applying the less than Best-Effort Scavenger service, thus allowing the coexistence of responsive Best-Effort traffic with high-bandwidth Scavenger flows

A slightly different implementation of the parallel data transfer can alternatively be based on the DS Assured Forwarding, which also can be used to implement a Guaranteed Rate service. Within one AF class a differentiation in terms of drop precedence can be applied to differently marked packets based on an implementation of Multiple Random Early Detection. At the ingress node of a DS domain a meter typically applies a token bucket mechanism such as the Two Rate Three Color Marker [28] proposed for the use with AF. This marker performs a marking of packets to be treated in the core with a low drop probability, if the traffic conforms to a committed information rate and with a high drop probability, if it exceeds this rate.

5.3. Multiple Flows: Wide Area Case

Our next experiments repeat those just described over the wide area network from ANL to LBNL: see Figure 4. Here, we used WFQ to configure our testbed with 55 Mb/s Premium traffic over the 60 Mb/s UBR VC between ANL and LBNL and 27 Mb/s Premium traffic within GARNET. Note that when congested this wide area Premium traffic configuration is a good approximation to priority queuing. (Only 27 Mb/s Premium traffic was allowed on GARNET in these particular experiments because of either extra traffic or a bad device on a fast Ethernet segment of the network that we were unable to control; in other experiments, we have successfully configured up to 45 Mb/s Premium.) Here, the background flow is initially allocated 24.3 Mb/s Premium bandwidth (that is, 90 percent of 27 Mb/s), the competing Best-Effort UDP flow operates at 50 Mb/s (1,250 byte packets every 200 μsecs), and two foreground flows are created: a 16 Mb/s flow (18 Mb/s reservation) at 37 secs and an 8 Mb/s

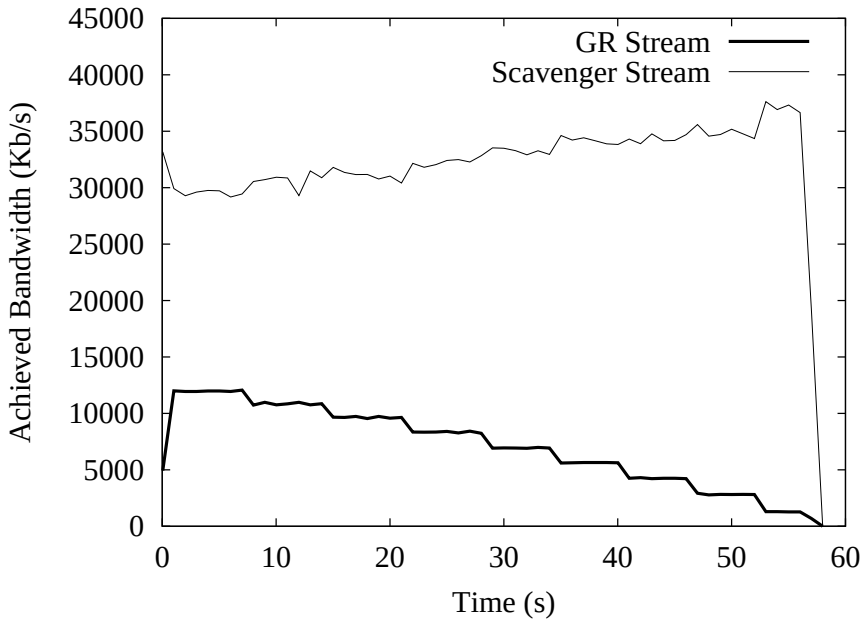

Figure 6. Performance achieved with a combination of a Guaranteed Rate service and a Scavenger service. We demonstrate that a bulk-transfer application with deadline is able to use a Guaranteed Rate service and in addition exploits unused Premium and Best-Effort capacity by a Scavenger service. See text for details.

flow (9 Mb/s reservation) at 94 secs.

As shown in Figure 8, the results obtained in the wide area are almost as good as in the local area. We attribute the somewhat more dynamic behavior during reservation changes to the fact that the kernel buffers associated with the bulk transfer socket take some time to empty. Hence, data is initially sent too rapidly for the updated router configuration, forcing packets to be dropped and TCP to go into slow-start mode. 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.

5.4. Evaluation of TCP Pacing

So far, we required the application to be instrumented to adapt to a given rate. In this section we introduce the use of traffic shaping as a vehicle to pace TCP throughput efficiently.

The actual throughput achieved by TCP applications depends on two main factors:

- The size of the advertised window determines the transmission rate of the transmitter (disregarding the congestion window). Using the socket API, the related window size can be influenced by explicitly setting the socket buffer size. The optimal socket buffer size should be equal to the bandwidth-delay product [46]. However, making the socket buffer too large might result in a reduced throughput because the sender might transmit more than the network is actually capable of handling, particularly during the slow-start process when TCP's self-clocking capability is not the dominant influence on bandwidth.

- The application has to provide data to the

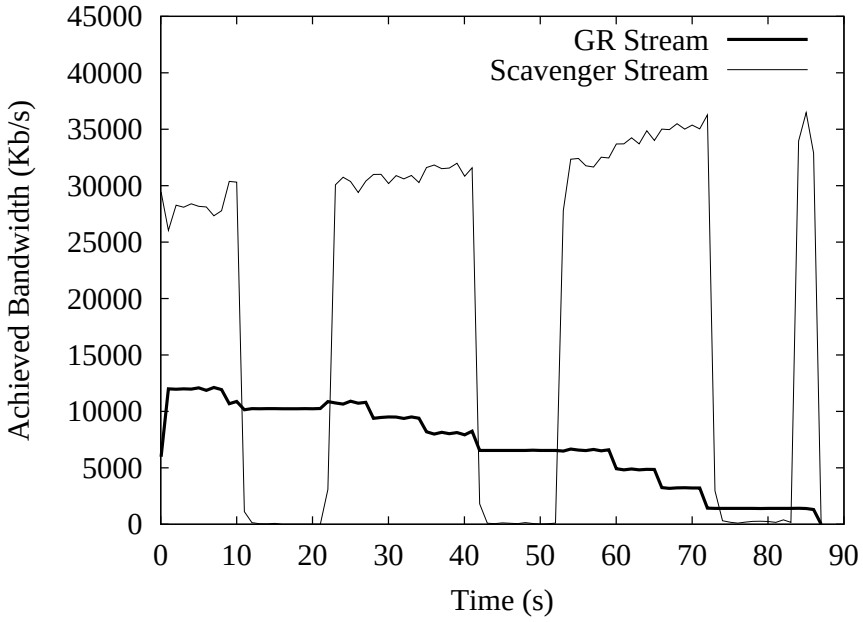

Figure 7. Performance achieved with a combination of a Guaranteed Rate service and a Scavenger service. While the Scavenger service reacts to periods of congestion, the Guaranteed Rate service remains stable. We see three periods of congestion: from 12-22, from 42-52, and from 72-82 seconds.

socket buffer that the TCP stack can actually fall into the so called steady-state. This ability often depends on the state of the local operating system, as data might be read from disk or the CPU is heavily used by other applications. The following section will address this issue.

There has been some discussion as to whether shaping of TCP traffic, i.e., TCP pacing, might increase fairness and throughput [34,2]. However, none of these studies was concerned with TCP flows using a virtual leased line, i.e. a Premium aggregate. Pacing TCP traffic in an environment offering a Guaranteed Rate service facilitates the simple use of network reservations even without the knowledge of the actual rate the application is writing data to the socket buffer. In the above experiments, we instrumented the application to adapt its socket buffer write frequency to the available background rate. In a scenario where TCP pacing is controlled by GARA's resource manager, the use of an oversized socket buffer leverages TCP's self clocking feature to control the speed of transmission. In coordinating a reservation with the shaping rate, packet drops can be avoided and a well-defined throughput can be established.

The following experiment is designed to demonstrate that shaping a TCP flow enables it to work smoothly with a Premium service without too much effort. We demonstrate two Premium TCP flows between Chicago (ANL) and California (LBNL) which try to exceed the rate they have reserved. This is a fairly likely scenario, since it is often hard for programmers to estimate the bandwidth their applications use. Also, we have already shown elsewhere [43,23] that applications that do not exceed their rate do not have a problem.

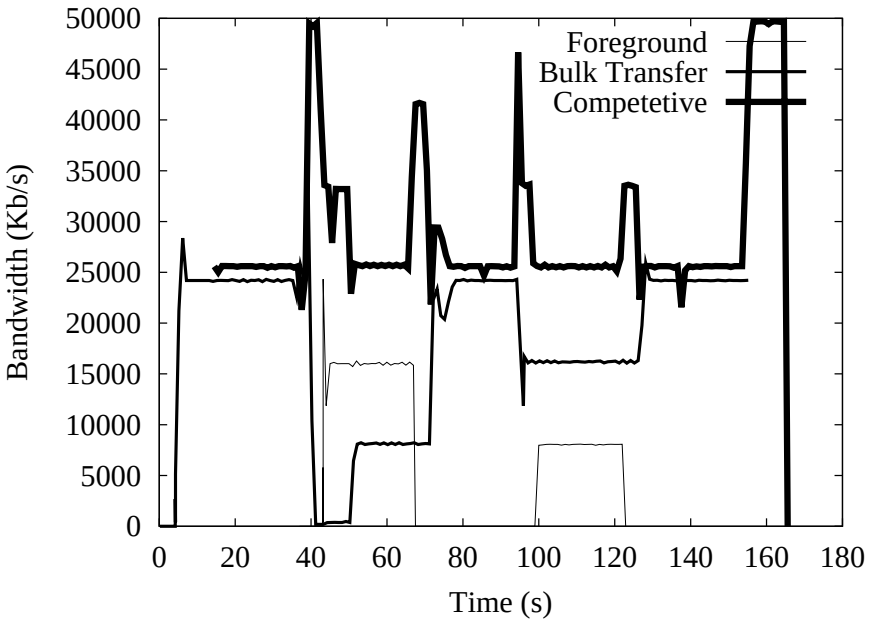

Figure 8. Performance achieved for a mixture of Premium and Best-Effort services on a wide area testbed. We demonstrate good performance even in the wide area. See text for details.

Specifically, we used a socket buffer size of 1MB and the round trip time is 75ms. Each flow ran at different times but are shown on the same graph. The application tried to write to the socket buffer at 64 Mb/s while it only made a reservation for 16 Mb/s. One of the flows is shaped to match the reservation bandwidth and therefore is paced to avoid packet drops. Figure 9 shows the achieved throughput for each of the flows. There are two things to notice in this figure. First, the shaped flow has a steady bandwidth at the reservation it made. Second, the unshaped flow has an unstable instantaneous bandwidth. Although it is not obvious from the graph, the average bandwidth of the flow is 9440 Kb/s, which is significantly less than the reservation.

One might hope that using selective acknowledgements would eliminate the need for shaping. This is because SACK can recover from multiple packet losses roughly in one round trip time. However, as can be seen from Figure 10, this is not the case. In this Figure, we repeated the same experiment as in Figure 9, but with selective acknowledgements enabled. In this case, the instantaneous bandwidth varies much less, but the average bandwidth is still significantly less than the reservation, i.e. 11992 Kb/s compared to 14448 Kb/s. Even though SACK can recover from packet losses more easily, TCP still reacts to the dropped packets as if they imply network congestion.

The results demonstrate that bulk transfer operation can be controlled by TCP pacing in a Guaranteed Rate service. Without detailed knowledge about the rate the application is sending at, GARA can react to the current reservation state and provide a constant, smooth transfer rate. Of course, a convenient shaping buffer has to be available in the edge router. This, however, is a realistic assumption as commodity networking devices offer buffer space in the order of MBytes [43].

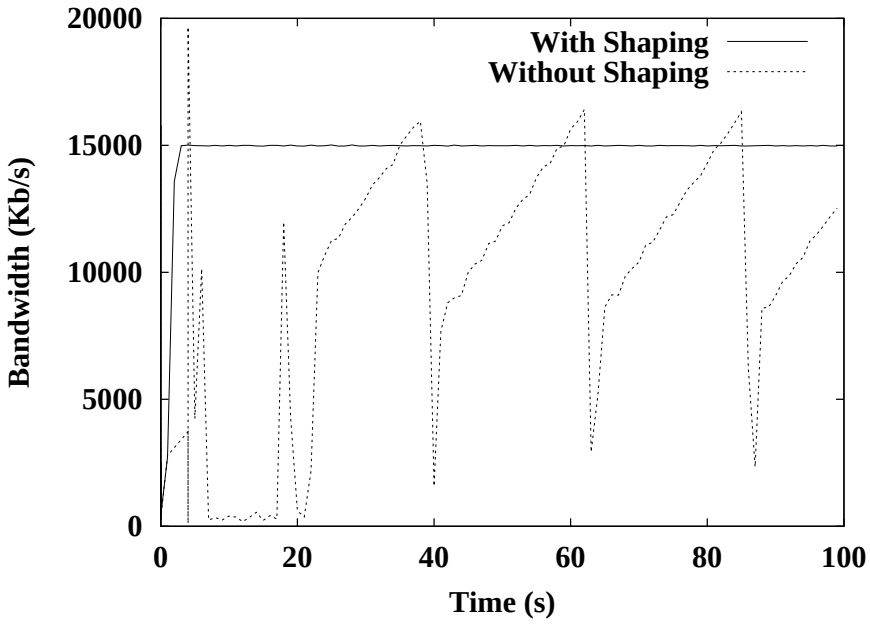

Figure 9. Achieved throughput for a Premium TCP flow exceeding the reservation. The SACK-option was disabled. The average achieved throughput was 9440 Kb/s without shaping and 14320 Kb/s when shaping was activated. See text for details.

The results also present the benefit of using SACK. While the experiment without using the SACK-option was oscillated due to packet loss, SACK could reduce its amplitude. However, the achieved throughput with SACK was significantly below the paced experiment.

5.5. Co-Reservation of CPU and Network

An important challenge addressed by GARA is the co-reservation of multiple resources: for example, network and CPU to ensure that a receiver can process incoming data. The experiment reported here demonstrates the ability of GARA to support such co-reservation. Specifically, we established a TCP flow and showed that we can maintain data transfer performance despite competing traffic on the network and competing computational load on the receiving host.

We conducted this experiment on GARNET and use the 100 Mb/s network as before, except that this time Premium traffic is configured to use up to 95 Mb/s. A TCP flow is started and network and CPU reservations and load are applied in various combinations.

1. A ttcp application is started without network congestion and without any reservation.
2. At 10 secs, an 80 Mb/s traffic generator is started. Because of network congestion, ttcp switches to the slow start feature and congestion control, with the result that ttcp performance drops precipitously and most available bandwidth is consumed by the competitive traffic.
3. At 40 secs, the TCP application creates an immediate network reservation through GARA. Performance increases dramatically.

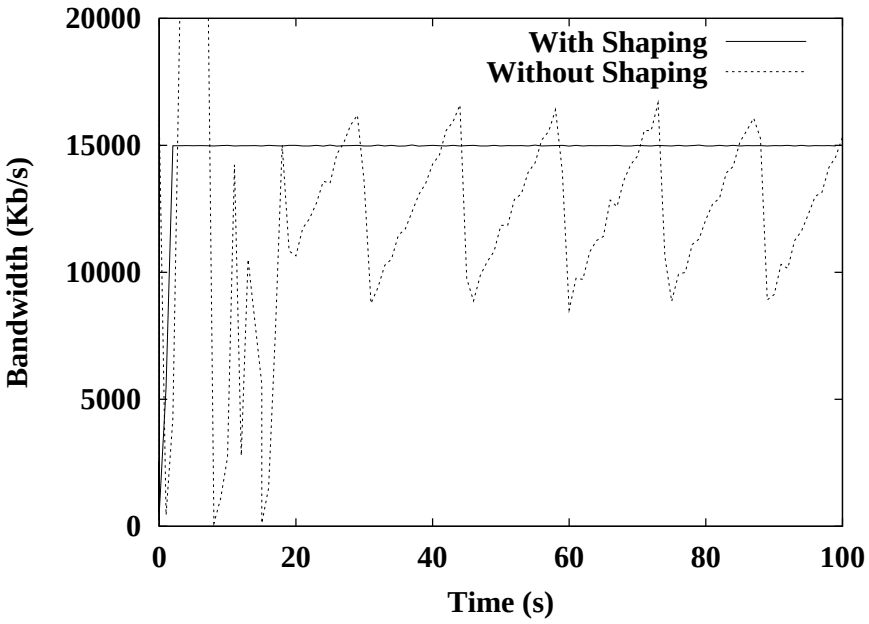

Figure 10. Achieved throughput for a Premium TCP flow exceeding the reservation. This time, the SACK-option was enabled. The average achieved throughput was 11992 Kb/s without shaping and 14448 Kb/s when shaping was activated. See text for details.

4. At 60 secs, a significant competing CPU load is imposed on the TCP receiver host. TCP throughput is significantly effected, due to contention for the CPU.

5. At 80 secs, we use GARA to reserve a significant amount of CPU for the receiving TCP process through the DSRT manager. The achieved rate increases immediately, although some variation remains due to the interval-based scheduling used by DSRT.

6. At 120 secs, we cancel the network reservation; TCP performance drops precipitously once again.

7. At 160 secs, we cancel the CPU reservation; this has little further impact on performance.

6. Policy in Multidomain Settings

We sketch an approach to expanding GARA to support more sophisticated policy enforcement, particularly in multi-domain settings.

6.1. General Approach

We assume the following system model. An end-to-end reservation may involve multiple resources located in different domains. Resource allocation decisions within a domain remain the responsibility of that domain; hence, end-to-end reservations must be authorized by all appropriate domains or by entities to which domains have delegated this authority.

Our policy approach is designed to support a flexible mix of policy options, for example:

- A domain may allocate resources on the basis of user identity. Such a policy may be appropriate in the case of unique resources

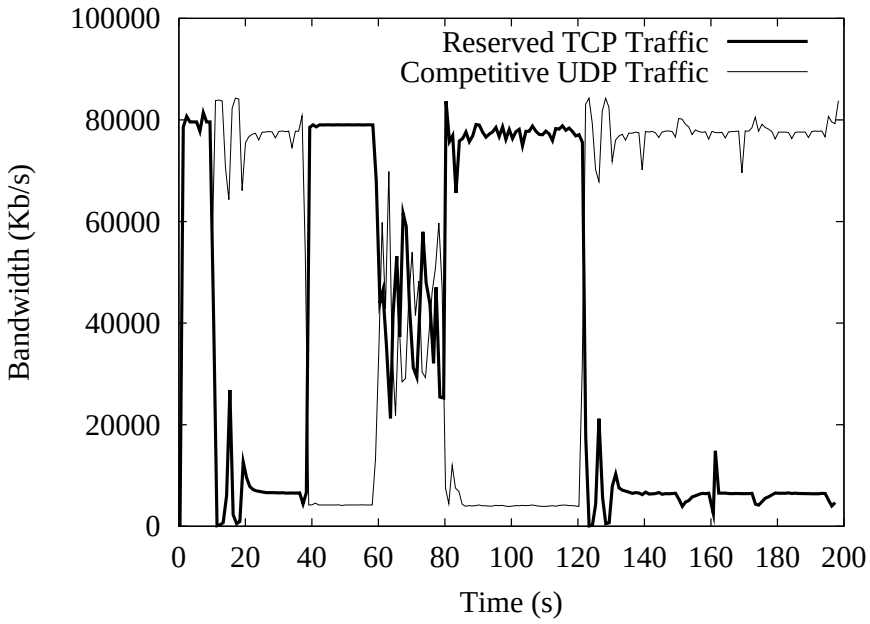

Figure 11. Performance achieved for a TCP flow in the presence of competing UDP traffic and host load, for various combinations of network and CPU reservation. We demonstrate GARA's ability to co-reserve multiple resource types.

for which users make distinct requests, e.g., supercomputers or specialized network resources such as a low-bandwidth outgoing connection.

- A domain may allocate resources in response to a request forwarded from another domain with which some agreement has been negotiated previously. For example, a transit service domain (e.g., ESnet in Figure 4) might negotiate an agreement to accept any allocation request forwarded from another DS domain, up to some SLA limit.
- A domain may allocate resources in response to a request authorized by some third party, such as a virtual organization with which the domain has negotiated an agreement previously [6]. This delegation of authorization allows a community to negotiate agreements with multiple domains in order to obtain control of some amount of Premium end-to-end bandwidth.

We anticipate multiple such authorization policies being active at one time. For example, in an environment such as that of Figure 4, a transit domain such as ESnet might support the following policies:

- Accept immediate reservations of Premium bandwidth from any domain with a previously negotiated SLA, subject to the constraint that no single request can be more than 100 Mb/s and the total requests from a domain cannot exceed its SLA.
- Accept immediate and advance reservation requests labeled as "HEP" if approved by a server operated by the high energy physics

community, up to limits and at times previously negotiated with that community.

We believe that authorization and authentication mechanisms provided in the Globus Toolkit provide a basis on which to explore these issues. The Akenti system [49] also provides important relevant technology. We provide a more detailed discussion of the handling of policies in such a distributed environment elsewhere [42].

6.2. Bulk Transfers in Multidomain Settings

So far our model for bulk transfers has assumed a single administrative domain. In a Grid environment, however, this assumption is of limited use, as virtual organizations typically consist of multiple administrative domains. In fact, whenever resources of different institutions are co-allocated, any request for network services passes at least three domains: the end-domains and one transient domain. Figure 12 illustrates this problem.

In a single-domain environment, one resource manager is able to provide a bulk-transfer class as described above. In a multi-domain environment, things are more complicated, because there can be many resource managers handling their own set of foreground and bulk transfer flows, and each such manager is a potential source of feedback to the application.

This problem can be addressed by introducing the abstraction of an aggregated end-to-end reservation or *core tunnel*. Users authorized to use this tunnel can then request portions of this aggregate bandwidth by contacting just the two end domains. The intermediate domains do not need to be contacted as long the total bandwidth remains less than the size of the tunnel. The importance of this aggregation is increased by the fact that in many Grids, multiple intermediate domains can be involved where aggregates are split into different egress points. By locating the bulk transfer within a single core tunnel we can perform the described adaption steps efficiently.

7. Related Work

The general problem of QoS implementation and management is receiving increased attention (see, e.g., [25]). However, there has been little work on the specific problems addressed in this paper, namely advance reservation and co-reservation of heterogeneous collections of resources for end-to-end QoS and the use of DS mechanisms to support flow types encountered in high-end applications.

Proposals for advance reservations typically employ cooperating servers that coordinate advance reservations along an end-to-end path [52, 17,14,27]. Techniques have been proposed for representing advance reservations, for balancing immediate and advance reservations [17], and for advance reservation of predictive flows [14]. However, this work has not addressed the co-reservation of resources of different types.

The concept of a bandwidth broker (similar to GARA's network resource manager) is due to Nichols and Jacobson [40]. The Internet 2 Qbone initiative and the related Bandwidth Broker Working Group are developing testbeds and requirements specifications and design approaches for bandwidth brokering approaches intended to scale to the Internet [48]. However, advance reservations do not form part of their design. Other groups have investigated the use of DS mechanisms (e.g., [54]) but not for multiple flow types. Hoo et al. [30] propose mechanisms for the secure negotiation of end-to-end reservations.

The co-reservation of multiple resource types has been investigated in the multimedia community: see, for example, [36,38,37]. However, these techniques are specialized to specific resource types.

8. Conclusions and Future Work

We have described a QoS architecture that supports immediate and advance reservation (and co-reservation) of multiple resource types; application-level monitoring and control of QoS behavior; and support for multiple concurrent flows with different characteristics. We have also

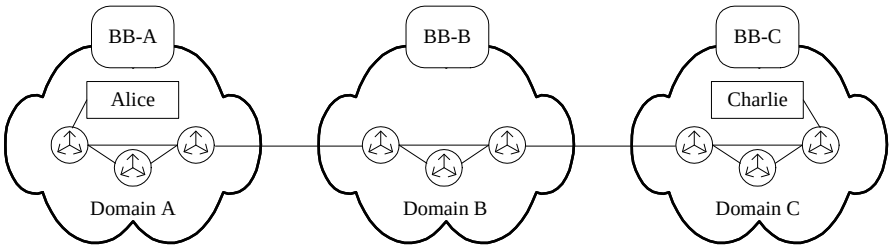

Figure 12. The multi-domain reservation problem. Alice needs to contact three BBs to make a network reservation from her computer in domain A to Charlie's computer in domain C.

described how this architecture can be realized in the context of differentiated service networks. We presented experimental results that demonstrate our ability to deliver QoS to multiple flows in local and wide area networks.

In future work we plan to improve and extend GARA in a variety of areas, including improved representation and implementation of policy, more sophisticated adaptation mechanisms (including real-time monitoring of network status), and more sophisticated co-reservation algorithms [11]. We also plan to extend our evaluation of GARA mechanisms to a wider range of applications and more complex networks. GARA mechanisms are being incorporated into the Open Grid Services Architecture-compliant [22] version 3.0 of the Globus Toolkit.

Acknowledgments

We gratefully acknowledge assistance provided by Rebecca Nitzan and Robert Olson with experimental studies. 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 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. M. Aeschlimann, P. Dinda, L. Kallivokas, J. Lopez, B. Lowekamp, and D. O'Hallaron, Preliminary report on the design of a framework for distributed visualization, *Proceedings of the Parallel and Distributed Processing Techniques and Applications Conference* (1999).
2. A. Aggarwal, S. Savage, and T. Anderson, Understanding the Performance of TCP Pacing, *Proceedings of IEEE Infocom* (2000) 1157–1165.
3. W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke, Data Management and Transfer in High-Performance Computational Grid Environments, *Parallel Computing* (2001).
4. H. Andrade, T. Kurc, A. Sussman, and J. Saltz, Active Proxy-G: Optimizing the Query Execution Process in the Grid, *Proceedings of SC* (2002).
5. W.Bethel, B.Tierney, J.Lee, D.Gunter, and

S.Lau, Using high-speed WANs and network data caches to enable remote and distributed visualization, *Proceedings of ACM/IEEE Supercomputing Conference* , (2000).

6. S. Blake, D. Black, M. Carlson, M. Davies, Z. Wang, and W. Weiss, An Architecture for Differentiated Services, *RFC 2475* (1998).
7. R. Braden, D. Clark, and S. Shenker, Integrated Services in the Internet Architecture: an Overview, *RFC 1633* (1994).
8. A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Data Sets, *Journal on Network and Computer Applications* **23** (2001) 187– 200.
9. H. Chu, and K. Nahrstedt, CPU Service Classes for Multimedia Applications, *Proceedings of IEEE Multimedia Computing and Systems* (1999).
10. K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, Grid Information Services for Distributed Resource Sharing, *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, (2001).
11. K. Czajkowski, I. Foster, and C. Kesselman, Co-allocation Services for Computational Grids, *Proceedings of the 8th IEEE Symposium on High Performance Distributed Computing* (1999).
12. B. Davie, A. Charny, J.C.R Bennett, K. Benson, J.Y. LeBoudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis, An Expedited Forwarding PHB (Per-Hop-Behavior), *RFC 3246* (2002).
13. T. DeFanti, and R. Stevens, Teleimmersion, [20] 131–156.
14. M. Degermark, T. Kohler, S. Pink, and O. Schelen, Advance Reservations for Predictive Service in the Internet, *ACM/Springer Journal of Multimedia Systems* , **5 (3)** (1997).
15. T. DeWitt, T. Gross, B. Lowekamp, N. Miller, P. Steenkiste, and J. Subhlok, ReMoS: A Resource Monitoring System for Network Aware Applications, *Technical Report, Carnegie Mellon University, CMU-CS-97-194* (1997).
16. N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merwe, A flexible model for resource management in virtual private networks, *ACM SIGCOMM* (1999).
17. D. Ferrari, A. Gupta, and G. Ventre, Distributed Advance Reservation of Real-Time Connections, *ACM/Springer Journal on Multimedia Systems* **5 (3)** (1997).
18. I. Foster, J. Insley, G. von Laszewski, C. Kesselman, and M. Thiebaux, Distance Visualization: Data Exploration on the Grid, *IEEE Computer* **32 (12)** (1999) 36–43.
19. I. Foster, and C. Kesselman, Globus: A Toolkit-Based Grid Architecture, [20] 259–278.
20. I. Foster and C. Kesselman (Eds.), The Grid: Blueprint for a Future Computing Infrastructure, *Morgan Kaufmann Publishers* (1999).
21. 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, *Proceedings of the International Workshop on Quality of Service* (1999) 27–36.
22. I. Foster, C. Kesselman, J. Nick, and S. Tuecke, The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, `www.globus.org/research/papers/physiology.pdf` (2002).
23. I. Foster, A. Roy, V. Sander, and L. Winkler, End-to-End Quality of Service for High-End Applications, *Technical Report, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne,* `www.mcs.anl.gov/qos/end_to_end.pdf` (1999).
24. E. Frécon, C. Greenhalgh, and M. Stenius, The DiveBone - An Application-Level Network Architecture for Internet-Based CVEs, *Symposium on Virtual Reality Software and Technology* (1999).
25. R. Guérin, and H. Schulzrinne, Network Quality of Service, [20] 479–503.
26. D. Gunter, B. Tierney, K. Jackson, J. Lee, and M. Stoufer, Dynamic Monitoring of

High-Performance Distributed Applications, *Proceedings of the 11th IEEE Symposium on High Performance Distributed Computing* (2002).
27. A. Hafid, G. Bochmann, and R. Dssouli, A Quality of Service negotiation Approach with Future Reservations (NAFUR): A Detailed Study, *Computer Networks and ISDN Systems* **30 (8)** (1998).
28. J. Heinanen, and R. Guérin, A Two Rate Three Color Marker, *RFC 2698* (1999).
29. J. Heinanen, T. Finland, F. Baker, W. Weiss, and J. Wroclawski, Assured Forwarding PHB Group, *RFC 2597* (1999).
30. G. Hoo, K. Jackson, and W. Johnston, Design of the STARS Network QoS Reservation System, *Technical Report, Lawrence Berkeley National Laboratory* (2000).
31. G. Hoo, W. Johnston, I. Foster, and A. Roy, QoS as middleware: Bandwidth broker system design, *Technical Report LBNL* (1999).
32. W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, and K. Stockinger, Data Management in an International Data Grid Project, *IEEE/ACM International Workshop on Grid Computing* (2000).
33. W. Johnston, J. Guojun, G. Hoo, C. Larsen, J. Lee, B. Tierney, and M. Thompson, Distributed environments for large data-objects: Broadband networks and a new view of high performance large scale storage-based applications, *Proceedings of Internetworking* (1996).
34. J. Kulik, R. Coulter, D. Rockwell, and C. Partridge, Paced TCP for High Delay-Bandwidth Networks, *Proceedings of the Workshop on Satellite-Based Information Systems (WOSBIS)* (1999).
35. S. Machiraju, M. Seshadri, and I. Stoica, A Scalable and Robust Solution for Bandwidth Allocation, *International Workshop on Quality of Service* (2002).
36. A. Mehra, A. Indiresan, and K. Shin, Structuring Communication Software for Quality-of-Service Guarantees, *Proceedings of the 17th Real-Time Systems Symposium* (1996).
37. K. Nahrstedt, H. Chu, and S. Narayan, QoS-aware Resource Management for Distributed Multimedia Applications, *Journal on High-Speed Networking* (1998).
38. K. Nahrstedt, and J. M. Smith, Design, Implementation and Experiences of the OMEGA End-Point Architecture, *IEEE JSAC, Special Issue on Distributed Multimedia Systems and Technology* **14 (7)** (1996) 1263–1279.
39. K. Nichols, S. Blake, F. Baker, and D. Black, Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers, *RFC 2474* (1998).
40. K. Nichols, V. Jacobson, and L. Zhang, A Two-bit Differentiated Services Architecture for the Internet, *RFC 2638* (1999).
41. E. Rosen, R. Callon, and A. Viswanathan, Multiprotocol label switching architecture, *RFC 3031* (2001).
42. V. Sander, W. A. Adamson, I. Foster, and A. Roy, End-to-End Provision of Policy Information for Network QoS, *Proceedings of the 10th IEEE Symposium on High Performance Distributed Computing* (2001).
43. V. Sander, I. Foster, A. Roy, and L. Winkler, A Differentiated Services Implementation for High-Performance TCP Flows, *Terena Networking Conference* (2000).
44. Q. Snell, M. Clement, D. Jackson, and C. Gregory, The Performance Impact of Advance Reservation Meta-scheduling, *IPDPS Workshop, Job Scheduling Strategies for Parallel Processing (JSSPP), Springer-Verlag LNCS 1911* (2000).
45. P. Steenkiste, Adaptation Models for Network-Aware Distributed Computations, *Proceedings of CANPC* (1999).
46. W. Stevens, TCP/IP Illustrated, Vol. 1 The Protocols, *Addison-Wesley* (1997).
47. B. Teitelbaum, Future Priorities for Internet2 QoS, `www.internet2.edu/qos/wg/papers/qosFuture01.pdf` (2001).
48. B. Teitelbaum, S. Hares, L. Dunn, V. Narayan, R. Neilson, and F. Reichmeyer, Internet2 QBone - Building a Testbed for Differentiated Services, *IEEE Network* **13 (5)** (1999).
49. M. Thompson, W. Johnston, S. Mudumbai, G. Hoo, K. Jackson, and A. Essiari,

Certificate-based Access Control for Widely Distributed Resources, *Proceedings of the 8th Usenix Security Symposium* (1999).
50. B. Tierney, W. Johnston, L. Chen, H. Herzog, G. Hoo, G. Jin, and J. Lee, Distributed Parallel Data Storage Systems: A Scalable Approach to High Speed Image Servers, *Proceedings of ACM Multimedia* (1994).
51. S. Vegesna (Eds.), IP Quality of Service, *Cisco Press* (2001).
52. L.C. Wolf, and R. Steinmetz, Concepts for Reservation in Advance, *Kluwer Journal on Multimedia Tools and Applications* **4 (3)** (1997).
53. J. Wroclawski, The Use of RSVP with IETF Integrated Services, *RFC 2210* (1997).
54. I. Yeom, and A. L. Narasimha Reddy, Modeling TCP Behavior in a Differentiated-Services Network, *Technical Report, TAMU ECE* (1999).