Chapter 23

GARA: A UNIFORM QUALITY OF SERVICE ARCHITECTURE

Alain Roy[1] and Volker Sander[2]
[1]*Department of Computer Science, University of Wisconsin-Madison*
[2]*Central Institute for Applied Mathematics, Forschungszentrum Jülich GmbH*

Abstract Many Grid applications, such as interactive and collaborative environments, can benefit from guarantees for resource performance or quality of service (QoS). Although QoS mechanisms have been developed for different types of resources, they are often difficult to use together because they have different semantics and interfaces. Moreover, many of them do not allow QoS requests to be made in advance of when they are needed. In this chapter, we describe GARA, which is a modular and extensible QoS architecture that allows users to make advance reservations for different types of QoS. We also describe our implementation of network QoS in detail.

1. INTRODUCTION

Many computing applications demonstrate increasingly voracious appetites, consuming ever more resources. From USENET to the spread of the World Wide Web to peer to peer file sharing, the demand for bandwidth on the Internet has been steadily increasing. Similarly, scientific programs used to measure their speed in megaflops, but now strive for teraflops and process terabytes instead of gigabytes [FK99b].

Just as data seems to expand to fill any size hard drive one can buy, today's most demanding applications strain the capacities of the networks, computers, and storage devices they use. When these applications must share their resources with other applications, they may be unable to perform to the satisfaction of their users. The problem is twofold: the resources are limited and the amount of a resource available to a particular application fluctuates depending on conditions beyond its control.

If an application does not have enough resources available to meet its performance needs, there are only two general solutions: the capacity of the resources

available to the application can be increased (such as buying more bandwidth), or the need for the resource can be decreased (such as decreasing the resolution of a streaming video). Sometimes resources have sufficient capacity for one application, but the actual capacity available to that application fluctuates because the resource is being shared with other applications. The most common example of this is a network, which is almost always shared between multiple applications. If we have such a shared resource and we cannot reliably have constant and sufficient service from it, there are two general strategies we can use. First, an application can adapt to the amount that is available. For example, a video streaming application may decrease the resolution of the video it sends when less bandwidth is available. Second, the resource may provide a guarantee that it will provide a certain quality, such as an upper boundary for the end-to-end delay, to the application. When a resource is able to offer such a guarantee, it is said to offer *quality of service*, or *QoS*.

While some applications are capable of easily adapting or may need only a single type of QoS, such as network QoS, others are very demanding and run in complex environments. They may require combinations of several types of QoS including network, CPU, and storage. Managing multiple resources with QoS can be difficult for applications because each type of QoS is typically controlled by a completely different system with different interfaces, capabilities, and behavior. Yet this management is important, because without combining different types of QoS, some applications may fail to operate well enough to meet users' expectations.

Additionally, applications often need to be scheduled. Sometimes an application needs to run at a particular time, perhaps to perform a demonstration or to be coordinated with some other activity. Other times, it is merely necessary to find a time when different QoS constraints can be simultaneously satisfied. In these examples, it is advantageous to be able to schedule the reservations for QoS in advance of when they are needed. We call these advance reservations. More precisely, an advance reservation is obtained through a process of negotiating a possibly limited or restricted delegation of a particular resource capability from the resource owner to the requester over a defined time interval.

To address this demand, we believe that there must be a resource management framework that is capable of providing a uniform interface to advance reservations for different types of QoS. To fill this need, we have developed such an architecture, the General-purpose Architecture for Reservation and Allocation (GARA) to allow demanding applications to easily manage quality of service for the various resources used by the application. GARA is a modular and extensible QoS system architecture that integrates different QoS mechanisms.

2. A UNIFIED ARCHITECTURE FOR QUALITY OF SERVICE

GARA provides a uniform mechanism for programmers to request QoS for different types of QoS. Once such uniform mechanisms are in place, it simplifies life for more than just the application programmer. It becomes possible to easily create higher-level services that can manage multiple simultaneous QoS requests on behalf of users. Perhaps the most important reason for having a uniform architecture for QoS is that it allows for relatively easy expansion of the services provided to users. As we will see below, GARA has a layered architecture that allows developers to easily add new QoS providers.

GARA's uniform architecture allows it to be easily used and extended by Grid users and developers. GARA has four key features:

- A uniform interface makes it easy to build services on top of GARA that provide new features to end-users, such as the ability to make coordinated reservations, or co-reservations.
- The ability to request advance reservations, in order to schedule applications against other constraints, or in order to find a time when all the QoS constraints will be able to be simultaneously met in the future.
- A layered architecture that allows for easy extensions as new QoS reservation mechanisms become available. For example, a graphical application that makes CPU and network reservations can easily add reservations for graphic pipelines if that ability is added to the lower layers of GARA. It is easy to add to the lower layers, and it does not require deep understanding of the higher layers in order to do so.
- GARA operates in a Grid infrastructure that includes a security infrastructure so that all reservation requests are securely authenticated and authorized. Security is an important aspect for a system that allows reservations, yet many QoS systems do not provide security. The Grid infrastructure that GARA currently uses is Globus.

2.1 Architecture

2.1.1 A Generic Framework for Advance Reservation in Grid Environments

GARA has a four-layer architecture, as illustrated in Figure 23.1.

The layer that most programmers would use is the *GARA layer*, which provides uniform remote access via the GARA Application Programmers Interface (API). This layer provides three essential services. First, it allows reservation requests to be described in a simple, uniform way. Second, when a

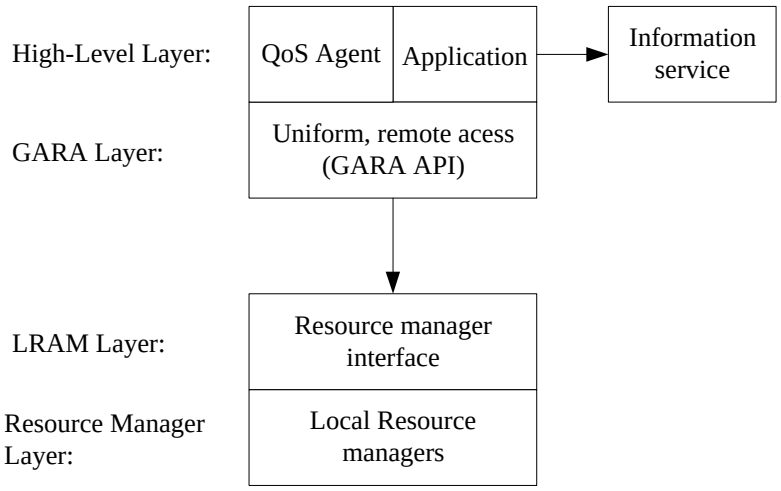

Figure 23.1. GARA's four-layer architecture. Applications and higher-level services use the GARA API, which communicate securely with the local resource layer, which in turn communicates with resource managers. Applications also communicate with an information service to find out information about resources for which they can make reservations.

reservation has been granted, it is described by a unique, data structure called a handle that can be stored, transmitted, and used to refer to the reservation. Third, it communicates with reservation services that may be located remotely or locally.

Applications also communicate with an information service that can inform them about likely reservations that can be made, and what to contact to make them. By combining resource reservation and allocation with the ability to search the information service, GARA offers a flexible framework for the construction of higher-level scheduling services.

The GARA layer communicates with the *LRAM layer*. The LRAM layer provides a resource manager interface that is responsible for authenticating and authorizing that the user is allowed to make a reservation. This layer is unaware of the specifics of the reservation, so it can only provide coarse authorization such as "Alice can make reservations", but not "Alice can only make reservations for bandwidths less than ten percent of the available bandwidth". That fine-grained authorization happens at a lower-level because it often depends on specifics of the available resource.

The LRAM layer is a "mostly uniform" interface to resource managers. It is not completely uniform because it is unnecessary: this layer provides a thin shim between the resource interface layer and the resource manager level beneath. It is the responsibility of this layer to translate all incoming requests so

that they can be presented to the resource managers that actually provide the QoS reservations.

The resource managers in the *resource manager layer* are responsible for tracking reservations and enforcing them by communicating with the lower-level resources, such as the network.

Instead of applications, there may be higher-level services in a *high-level layer*. These handle QoS requests for applications, often interacting with the information service and making multiple reservations at the same time. Such services are discussed in Section 3.

This four layer architecture allows for a uniform interface at the top, secure access to remote resources, and any number of QoS implementations.

2.1.2 Resource Reservations: A High-Level Interface for Grid Applications

Let us take an example of how a programmer may make a reservation for network bandwidth needed tomorrow afternoon. First, the program needs to make a list of the attributes needed for the reservation. GARA uses a text-based attribute-value representation of a reservation. The representation language we currently use—the Globus Resource Specification Language, or RSL [CFK+98b]—is schema-free, but GARA has some standard attributes. A typical reservation request may look like:

```
&(reservation-type=network)
 (start-time=953158862)
 (duration=3600)
 (endpoint-a=140.221.48.146)
 (endpoint-b=140.221.48.106)
 (bandwidth=150)}
```

The first three fields (reservation-type, start-time, and duration) are used for all reservations. The last three fields are unique to network reservations.

To request the reservation, the programmer does:

```
(error, handle) = reservation-create(resource-name, resv-desc);
```

Assuming there is no error, the reservation has been made. It can be queried at any time to find out the status of the reservation:

```
(error, status) = reservation-status(handle);
```

There is also an asynchronous event delivery service that can inform a program about reservation related events. These events are sent by the resource manager. Example events are a notification that a reservation time has begun or that an application is sending data faster than the reservation allows [FRS00].

When a program is ready to use a reservation, it sometimes needs to inform GARA of the information that was not previously available. For example, a network reservation needs to provide the port numbers used by the TCP or UDP connection so that the network routers can provide QoS guarantees, but these port numbers are not known in advance. Providing this information is known as binding the reservation:

```
bind_params = "(endpoint-a-port=1234)"
            + "(endpoint-b-port=5678)";
error = reservation-bind(handle, bind_params);
```

When the program is done with a reservation, it can be canceled:

```
error = reservation-cancel(handle);
```

Note that the information passed within a bind request is always related to the requested type of service. GARA uses RSL to provide a generic interface. As much as possible, these parameters are kept consistent in GARA, but they must change to reflect the underlying properties of the QoS. Beyond this difference though, GARA present a uniform interface to the underlying QoS providers.

2.1.3 Resource Managers: Service Provisioning for Grid Resources

A *resource manager* translates requests for QoS into actions that need to be taken to ensure that the QoS is provided to the application. For instance, a resource manager may configure a router to ensure that an application receives the bandwidth that it requested.

GARA was designed to make it easy to integrate new resource managers written by other people, but we also created several resource managers just for use in GARA. For GARA, we created a network QoS resource manager that uses differentiated services, a prototype disk space QoS resource manager, and a CPU QoS resource manager that uses process priorities. We also created two hybrid resource managers: one interacts with the Dynamic Soft Real-Time (DSRT) CPU scheduler [CN99] and adds advance reservations, another interacts with the Portable Batch System (PBS) which already provides advance reservations. A collaborator created a resource manager for graphic pipelines. Although we worked with a number of resource managers, most of our focus was on the network resource manager.

To be used in GARA, resource managers need to have a few common features:

- *Advance Reservations.* Each resource manager must support advance reservations. If a resource manager does not support advance reservations, support can be added by using a hybrid resource manager on top

of the resource manager, similar to the DSRT example mentioned above. Within GARA, we developed a simple but effective *slot table manager* to manage reservations that are considered as slots in time. Each slot represents a single capacity delegation as a "slot" of time. These slot tables can be used by any resource manager to keep track of reservations. In addition to the provision of basic slot table operations such as creating, modifying, and deleting an entry, the manager can also deliver asynchronous events when a reservation begins or ends. Therefore, it offers an implementation framework for implementing advanced notification services as described above and can be reused in different resource managers.

- *Interaction with Resource.* Each resource manager needs to interact with the underlying resource in order to enforce the QoS. If the resource manager does not have complete control over the QoS, then reservations cannot be guaranteed.

- *External Interface.* Services provided by resource managers need to be accessed. Because GARA incorporates the interface of resource managers into a Grid resource management framework, it depends on the ability to interface directly with the resource manager. Note that GARA was implemented within the Globus framework which provides user delegation. It therefore knows which user is accessing a resource manager, and all interaction happens as that user.

Note that resource managers do not necessarily need to provide authentication or remote access, since that is provided through the higher levels in GARA. However, because GARA understands users and uses delegation when users authenticate, resource managers can do additional authentication.

3. CO-RESERVATIONS

Most QoS research has concentrated on single types of reservations, whether network reservations [FV90], CPU reservations [LRM96], or disk reservations [MNO+96]. However, it is often important to use different reservations at the same time. When multiple reservations are made at the same time, we call them coordinated reservations, or *co-reservations*.

Consider, for example, the scientific visualization application shown in Figure 23.2. Here we have an application reading experimental results from disk, rendering the results by creating lists of polygons, and sending the results to a remote computer which then visualizes the results. If the entire system is manually reserved to be used by the application alone, perhaps by a phone call to a system administrator, then no QoS mechanism is necessary. However, if we are using shared systems, any portion of the system could experience con-

tention, slowing down the scientific visualization. In particular we could have contention for:

- the disk system where the experimental results are stored,
- the CPU doing the rendering,
- the network used for sending the rendered data,
- the CPU displaying the final results.

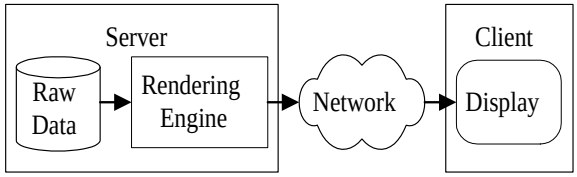

Figure 23.2. An application that could benefit from co-reservation. An example of an application that could benefit from co-reservation. Because the reservation pipeline uses several different potentially shared resources, it is likely to be beneficial for the application to make a reservation for each resource: disk, graphic pipeline, computer, display, and network.

Any one or a combination of these systems could require the use of QoS. We need to make reservations for each system to ensure that everything works smoothly when we cannot predict what contention will occur in the future.

Figure 23.3 shows a concrete example of the usefulness of co-reservation. In this example, an application is attempting to send data at 80 Mb/s using TCP. Because the application is sending at a high rate, it may delay in sending data if the CPU is busy. Because of TCP's sliding window mechanism, this may result is significantly lower bandwidth. In the experiment, the application experienced two types of congestion. First, there was network congestion beginning at about time 15 and continuing to the end of the experiment. A network reservation was made at time 40 to request for an appropriate bandwidth. Second, there was contention for the CPU at about time 60 and continuing for the rest of the experiment. A CPU reservation was made at time 80 to correct for this. From time 80 to 120, both reservations were active, and the application was able to send data at its full rate. co-reservation.

Although the application shown in Figure 23.3 was an experiment and not performed with a real application, it reinforces our point that it is often important to combine different types of reservations.

Because GARA has a uniform interface to multiple types of underlying reservation systems, it is fairly easy to build co-reservation agents that manage the multiple reservations on behalf of a user. We have built such co-reservation agents, and they are described in [Roy01].

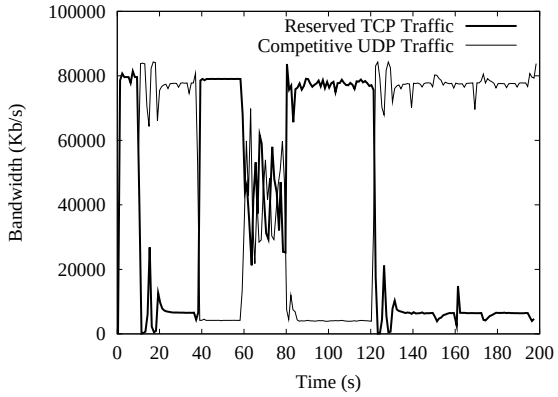

Figure 23.3. Combining DSRT and differentiated services reservations.

4. NETWORK RESERVATIONS

In a Grid environment, networks are essential to the smooth operation of the Grid, and they are also often the most shared resources. Therefore, when we developed GARA we spent considerable effort in ensuring that GARA could effectively manage advance reservations and QoS for applications that made demanding use of networks.

Grid applications have a wide-variety of network QoS needs which are much more challenging than other network QoS applications such as voice over IP. This is because Grid applications use a wide variety of network flows, including low-bandwidth but low-latency control flows, high-bandwidth and low-latency data transfers for applications such as visualization, and high-bandwidth but not necessarily low-latency data transfers for moving large amounts of data for analysis and other purposes. This combination of types of network flow places strong requirements on the network QoS layers. When we examined the needs of various applications [FK99b] we saw the need to support two basic services: a premium service offering a low-delay virtual leased line and a guaranteed rate service. Because of the wide variety and complexity of network demands coupled with the requirement to incorporate the resource "network" into our Grid resource management framework, GARA has a flexible resource manager that addresses the particular requirements of emerging Grid applications in IP-based networks.

Initially, our efforts focused on providing network QoS within a single network domain, which considerably simplified the problem. Later, we investigated providing network QoS between multiple network domains. All of these efforts were implemented in GARA at the resource manager level.

4.1 Single Domain Network Reservations

We initially built a GARA resource manager that could provide reservations within a single network domain. This facilitated a rapid prototyping, because it is possible to give it access to all of the relevant network resources in order enforce the QoS.

The first problem we considered was the mechanism to use to implement the network QoS. QoS in an Internet Protocol (IP) network can be provided in different ways. Over the years, two primary approaches have been used and are exemplified by two standards published by the Internet Engineering Task Force's (IETF). One of these is Integrated Services [Wro97, BCS94] which provides service guarantees based on a flow-based packet differentiation in each router, and the other is Differentiated Services, or diffserv, which differentiates only the treatment of classes of packets called aggregates instead of individual reservations. Integrated Services has been largely abandoned in favor of diffserv, and we do not discuss it further here.

The diffserv architecture [BBC+98] is a reaction to the scalability problems of the flow-based approach of the Integrated Services architecture, and does not provide reservations. Instead, packets are identified by simple markings in the type of service field of the IP-header [NBBB98] that indicate how routers should treat the packets. In the core of the network, routers need not to determine which flow a packet is part of, only which aggregate behavior they should apply. In this scenario, one needs to decide which packets get marked–this is how a higher-level service can provide reservations. To do this, a particular resource manager called a *bandwidth broker* is used. A bandwidth broker is a middleware service which controls and facilitates the dynamic access to network services of a particular administrative domain. Our goal for GARA was to design and implement a resource manager which fulfills the functions of a bandwidth broker.

Diffserv allows packets to be marked either by applications or by the first router that receives the packets—the edge router. If applications are allowed to mark packets, QoS cannot be guaranteed, so GARA uses the edge routers to mark packets. In order to enforce the reservation, packets are only marked when they are "within profile"—that is, when the sender is sending within rate given to the reservation.

Core routers (those that are not on the edge) have an easier job because they do not need to identify packets that need marking, nor police packets to ensure they are within profile. Instead, core routers apply a particular packet treatment—called per-hop behavior (PHB)—based only on these markings. Currently, the Internet Engineering Task Force's Differentiated Services Working Group has specified a small set of PHBs [DCB+01, HFB+99].

GARA uses the Expedited Forwarding (EF) PHB, which is intended to be used for a high-priority service with little jitter and queuing delay. The exact definition of EF is rather technical, so to simplify, each router interface can be configured so that traffic marked for the EF aggregate is prioritized over all other packets. To prevent starvation, we have to limit the amount of data which is marked as EF. This admission control is the task of the GARA resource manager. Details of how EF is required to work are defined in [CBB+02]. We have found that when carefully used, EF can provide robust reservations.

In order to implement a premium service based on EF, GARA assumes that each output link is configured to identify EF packets and to prioritize them appropriately by applying priority queuing. Note that this potentially requires a configuration update of all routers in a domain. Fortunately, this only has to be done once. While the admission control procedure uses the slot table manager to respond to reservation requests, reservations also have to be instantiated and policed in the edge routers. GARA dynamically configures the packet classifier, the policer, and the packet marker to appropriately map packets to EF. Figure 23.4 illustrates this scenario. Packet classification is done based on the reservation attributes specified when a user made a reservation. When applied to the scenario we describe here, it is done based on the end-points (address and port number) and the protocol (TCP or UDP).

Once the packets have been classified, a so-called "token bucket" is used to ensure that during the time interval $[t_0, t_1]$ of length $T = t_1 - t_0$ the amount of data sent does not exceed $rT + b$ bytes. Here, r denotes the average rate the at which the token bucket operates and b represents the depth of the token bucket which allows some limited bursts. Packets which fulfill this constraint will be marked to belong to EF. To do this correctly, GARA identifies and configures the relevant edge router every time a reservation is activated.

Note that applications may have a difficult time staying within the limits of their reservations. While monitoring the policing function and providing feedback to the application is appropriate for UDP-based flows, this mechanism does not work well for TCP-based communication. In order to assist applications, a third mechanism, called traffic shaping, is used for the traffic entering the edge router. The idea is to shape the injected TCP-traffic that it injects a smooth rate that conforms to the reservation to the core network. By incorporating this with the relaxed configuration of the policing function, TCP-applications can effectively paced to use their reserved bandwidth. Details of work we have done with this can be found in [SF02].

In previous papers such as [SFRW00], we have described many experiments that demonstrate the details of implementing such schemes successfully. GARA follows a concept which we call the "easy-to-deploy" paradigm, that is, GARA's ability to provide network services to Grid applications does not rely on complex nor unrealistic assumptions. Based on the deployment of a

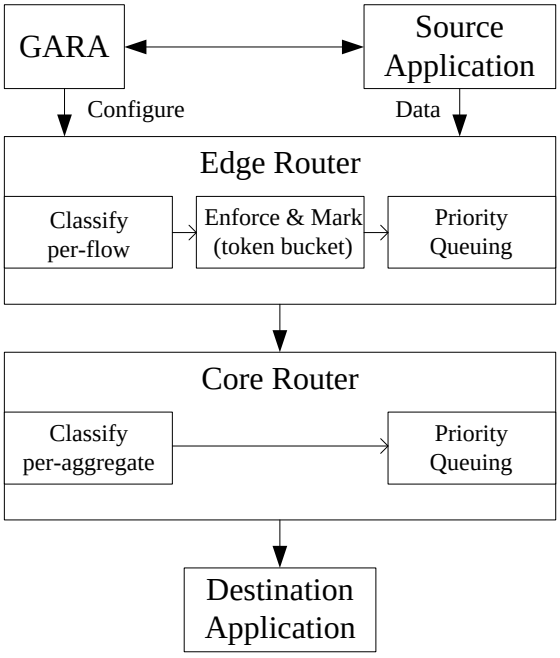

Figure 23.4. A simple network that shows how GARA uses diffserv. GARA configures edge routers to use classification, marking, and enforcement per-flow, and priority queuing is used in the core.

single prioritized PHB, GARA is able to provide a premium and a guaranteed rate service. Furthermore, we do not rely on changes of the transport protocol, nor specific advanced capabilities of the operating system, such as the support of traffic shaping. A comprehensive discussion can be found in [San03].

4.2 Multi-Domain Network Reservations

As mentioned above, the GARA network resource manager that implements network QoS is an example of what is commonly known as a bandwidth broker. Because of the fact that end-to-end guarantees in Grid environments are likely to happen in complex network environments where multiple independent administrative organizations are responsible for the operation of subparts of the network, it is very unlikely that a single bandwidth broker will control more than one administrative domain. Instead, each administrative domain wishes to have control over their resources and will thus operate its own policy decision point.

Therefore, bandwidth brokers must interact with other bandwidth brokers. A network reservation for traffic traversing multiple domains must obtain mul-

tiple network reservations, as shown in Figure 23.5. Here, Alice wants to make a network reservation from her computer in *source domain* A to Charlie's computer in *destination domain* C. Somehow she needs to contact and negotiate a reservation with BB_A and BB_C as well as the *intermediate domain*, BB_B. We have experimented with two approaches to multi-domain reservations: co-reservation and chained bandwidth brokers.

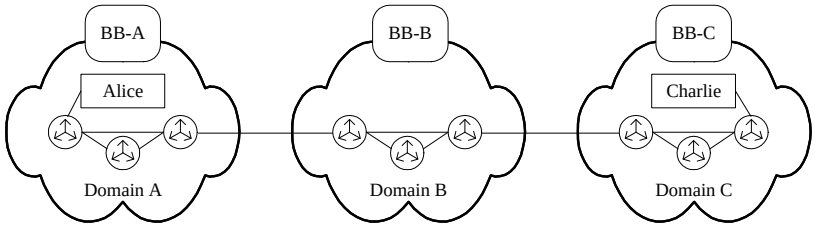

Figure 23.5. The multi-domain reservation problem. Alice needs to contact three bandwidth brokers (BB-A, BB-B, BB-C) to make a network reservation from her computer in domain A to Charlie's computer in domain C.

4.2.1 Using Co-Reservation

Alice, or an agent working on her behalf, can contact each bandwidth broker individually A positive response from every bandwidth broker indicates that Alice has an end-to-end reservation. However, there are two serious flaws with this methodology. First, it is difficult to scale since each bandwidth broker must know about (and be able to authenticate) Alice in order to perform authorization. Furthermore, if another user, Bob, makes an incomplete reservation, either maliciously or accidentally, he can interfere with Alice's reservation. An example of this type of bad reservation is illustrated in Figure 23.6.

The authorization problem could be solved if Alice could acquire some common credential issued by a community wide authorization server. GARA could interoperate with the Community Authorization Server (CAS) [PWF+02] of the Globus project to achieve this. However, the problem of incomplete reservations discouraged us from pursuing network co-reservations further.

4.2.2 Using Chained Bandwidth Brokers

The problems just noted are a motivation for the specification of an alternative approach, in which reservation requests are propagated between bandwidth brokers rather than all originating at the end domain. As shown in Figure 23.7, this means that Alice only contacts BB_A, which then propagates the reservation request to BB_B *only if* the reservation was accepted by BB_A. Similarly,

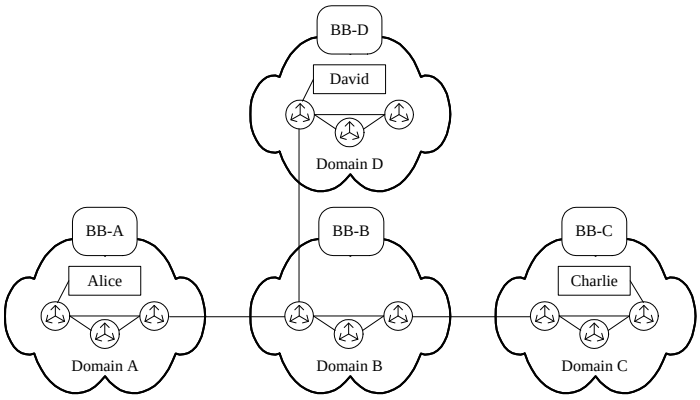

Figure 23.6. Consistency problem of source-domain-based signaling. David, a malicious user in domain D, makes a reservation in domains D and B, but fails to make a reservation in domain C, even though he will be sending his marked packets to Charlie in domain C. Domain C polices traffic based on traffic aggregates, not on individual users, so it cannot tell the difference between David's traffic and Alice's reserved traffic. Therefore, there will be more reserved traffic entering domain C than domain C expects, causing it to discard or downgrade the extra traffic and thereby affecting Alice's reservation.

BB_B contacts BB_C. With this solution, each bandwidth broker only needs to know about its neighboring bandwidth brokers, and all bandwidth brokers are always contacted. In addition to this chained signaling approach, Figure 23.7 also demonstrates the use of the GARA API (see Section 2.1.2) to couple a multi-domain network reservation with a CPU reservation in domain C.

With this chained signaling approach, the bandwidth broker interfaces not only with the high-level GARA interface for application, but also with its peer bandwidth brokers. The Internet2 community [ABC$^+$01] has proposed using a long term TCP connection to establish a stateful communication between peered bandwidth brokers. However, a reservation actuator accompanies reservations during their lifetime. Therefore, there is no need for a long term connection for individual requests. The abstraction of a traffic trunk is the resolution for these heterogeneous demands. While a traffic trunk represents a single reservation for end-domains, it represents the pieces of interest for transient domains: *core tunnels*. A core tunnel is an aggregated uni-directional reservation between the two end-domains. It connects the egress router of the source-domain with the ingress router of the destination-domain by means of the service request parameters. By introducing a traffic trunk for each core tunnel, a reservation actuator accompanies a core tunnel during its lifetime. It subscribes to events signaled by the peered domains and in doing so, it enforces

a TCP connection for all entities which have registered a callback which lifetime is related to the lifetime of the core tunnel. For static Service Level Agreements (SLAs), the proposed model conforms to the SIBBS model, because a static SLA is represented by a set of long term core tunnels. A comprehensive discussion on this approach can be found in [SAFR01, San03].

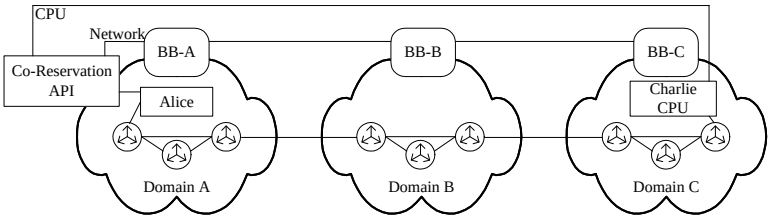

Figure 23.7. Multi-domain reservations with hop-by-hop-based signaling. Hop-by-hop-based signaling of QoS requests is done using an authenticated channel between peered bandwidth brokers along the downstream path to the destination.

4.2.3 Building Per-Domain Behaviors

The purpose of specifying PHBs for aggregates is to establish services. Because diffserv is used in domains in which the specific PHBs are applied, services are established by domains and are provided within domain boundaries. [NC01] defined this more precisely as a Per-Domain Behavior (PDB). It describes the expected treatment that an identifiable or target group of packets will receive from "edge-to-edge" of a diffserv domain. The creation of a core tunnel in transient domains can thus be interpreted as an agreement to serve the related aggregate with a particular service level, or PDB.

Earlier, we described GARA's slot table manager for performing its admission control task. Initially, we used it in a simplistic way, and used a single slot table for a whole domain. This solution limited the offered service to the achievable service of the link with the minimum QoS capability of the domain, that is, we assumed that all requests will flow through this particular link.

Later, we used a more advanced admission control procedure using the knowledge about the network topology and about the routing tables, which is able to identify the actual path of the request in the controlled domain. In this case, the service was not limited by the minimum link capability anymore. However, also this approach does have its limitations. In the fuzzy context of aggregate based scheduling it is hard to provision strict delay and jitter boundaries [CB00]. We therefore respected the ability to incorporate traffic engineering capabilities into the admission control procedure of GARA. In ex-

tending the interaction with the edge router by also controlling the use of the traffic engineering capabilities of the MultiProtocol Label Switching (MPLS) architecture [RVC01, RTF+01], GARA offers a flexible framework for service provisioning in transient domain. Applying network calculus [BT00, Bou96], a formal method for the worst-case analysis of the achievable network service, gives the opportunity to use this feature for the establishment of strong service guarantees [San03, Fid03].

5. AN IMPLEMENTATION FOR THE GLOBUS TOOLKIT

The current implementation of GARA uses the Globus Toolkit as a foundation. It was initially implemented as part of Globus 1.1.3, although a port to Globus 2.0 has been done by a third party.

The four layers of the GARA architecture shown in Figure 23.1 map closely to the layers of the Globus Toolkit. The GARA API, which resides in the remote access layer, corresponds closely with the Globus Resource Allocation Manager (GRAM) API [CFK+98b], which uses the Grid Security Infrastructure (GSI) for authentication (see Chapter 5). The Globus gatekeeper is responsible for authenticating and authorizing all GRAM and GARA interactions with a system. The LRAM layer and the local resource managers do not have exact analogues in the Globus Toolkit, but were implemented completely within the GARA framework.

Because the protocol for communication with the gatekepeer and the security mechanisms were already completely existing within the Globus Toolkit, we were able to easily leverage them without any loss of generality or flexibility in the overall architecture of GARA.

5.1 Security

Globus uses the Grid Security Infrastructure (GSI) [FKTT98]. The GSI normally uses public key cryptography. Users have private keys that they never share, and public keys (called certificates) that anyone can view. An important aspect of GSI is that it allows users to delegate credentials. To do this, a user can create a proxy certificate which has a new public and private key and is signed by the user's private key. However, it usually has a much shorter life time, generally on the order of twelve to twenty-four hours. This proxy certificate can then be used for authentication. If the proxy should happen to be compromised, it will useful for a much shorter time than the user's private key.

5.2 The Gatekeeper Protocol

GARA uses GSI to authenticate with the gatekeeper. After authentication, the gatekeeper passes the network connection to another program called the GARA service. This GARA service uses the Local Resource Manager (LRAM) API to interact with the local resource managers. Each GARA API call is a transaction with the gatekeeper, so each call benefits from the security and remote access capability.

The GARA API allows users to request callbacks that inform the user when changes to the reservation occur. These do not use the gatekeeper for callbacks, but retain the connection originally opened to the gatekeeper, but redirected to another program that provides the callbacks.

5.3 Mapping of Service Requests to Resource Managers

As mentioned above, the GARA service uses the LRAM API. This is similar to the GARA API, but it does not provide remote access or security. It does provide an abstract interface to the resource managers so that the GARA service does not require intimate knowledge of different resource managers. Instead, the LRAM API knows the details of speaking to the resource managers.

The LRAM is implemented as a series of libraries that can be used to communicate with different resource managers. While it was written in C, it is essentially an object-oriented framework that allows for abstract interfaces to a variety of implementations.

6. FUTURE WORK

Although GARA has been demonstrated to be an effective system [SFRW00, FRS00, San03, Roy01], it is a first-generation architecture, and there are important improvements we are planning for GARA.

To improve GARA's functionality in Grid environments, we will extend its uniform API, particularly to include a two-phase commit protocol, which is essential for reliability. We also intend to move GARA to the Open Grid Services Architecture (OGSA) [FKNT02, TCF+03]. For a detailed discussion about this convergence refer to [CFK+02].

GARA needs a better mechanism for helping users find reservations. Currently GARA publishes information about what reservations may be available in an information service, and it is up to the user to ask for a reservation that may work. GARA can only respond with "yes" or "no" when a request is made. A more effective approach is the sort used by *ClassAd matchmaking* [RLS98]. Such a system would allow users to say, "I need 10-15 megabits per second for an hour tomorrow afternoon".

Enhancing GARA's resource managers is driven by user demand. The improvement of the network resource manager is still an ongoing effort. The Path Allocation in Backbone networks (PAB) project [SIF03] funded by the German Research Network (DFN) and the Federal Ministry of Education and Research (BMBF) is developing an optimized admission control procedure. The embedded advanced traffic engineering capabilities can be optimized based on an emerging simulation tool. We intend to integrate the results within GARA's admission control procedures.

Extending GARA's reach to other types of QoS, particularly disk space reservations would be very useful. Our prototype disk space resource manager was sufficient to show that it was interesting, and we believe that interacting with a system such as NeST (see Chapter 21) would work well.

7. CONCLUSIONS

GARA is an architecture that provides a uniform interface to varying types of QoS, and allows users to make advance reservations. In our experience, GARA has provided a useful framework in which to experiment with different types of QoS. In particular, we have experimented heavily with network QoS, but have also investigated providing reservations for computers, CPU time, disk space, and graphic pipelines. We believe that GARA is a promising platform for future investigations into quality of service.

For those interested in further discussion of this topic, please see Chapter 8 and [Roy01, San03].