Network Support for Application-Oriented QoS

Prashant Chandra, Allan L. Fisher, Corey Kosak and Peter Steenkiste

School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
{prashant+,alf+,kosak+,prs+}@cs.cmu.edu

Abstract

This paper addresses a dilemma raised by recent advances in networking technology, which provide support both for a rich variety of qualities of service, and for applications that connect many endpoints. Together, these features encourage the development of complex multi-party applications that use a diverse set of data types. This raises a two-fold problem: how do application designers choose and specify the many quality of service parameters that drive the ultimate performance of their applications; and how does the network efficiently manage its resources to support such a rich application mix?

Our approach to this problem is to allow applications to be built around value-added services that encapsulate a variety of simpler resources. This enables both the specification of quality of service (QoS) in terms meaningful to applications, and global optimization of resource allocation across multiple streams and data types. In this paper, we present a network architecture and a preliminary implementation that explicitly support the notion of application-oriented QoS for complex network services. The key concept is that of service brokers, *which applications and service providers use to identify the network resources needed to meet QoS and cost objectives. Service brokers can incorporate a detailed understanding of an application domain, allowing them to make intelligent tradeoffs and to interact with applications and service providers at a high level. They can be hierarchical, in the sense that one broker can invoke the services of another broker. Finally, they provide the ability to deal with heterogeneous networks and hierarchical resource management.*

1 Introduction

While for several decades, computer networks have been primarily used to move data using programs such as ftp and telnet, they are increasingly being used for more complex tasks (e.g., information retrieval), or as a tool in support of interpersonal communication (e.g., video conferencing). We expect this trend to continue, and the Internet to become a sophisticated infrastructure that can deliver a wide range of complex services. While today's network applications view the network as providing simple bit pipes, future multi-party applications will need more complex multi-point to multi-point connectivity for the transfer of a variety of data types with different characteristics. We also envision the delivery of "value-added" services that build on lower-level services to deliver not only data communication, but also computation and storage functionality in an integrated fashion. Example services are reliable multicast, intelligent caching, video/audio transcoding and mixing, translation of different languages, and database access.

In the last few years, the networking community has been developing support for multiple service classes with different "quality of service" (QoS) properties. Different classes are needed to support applications with specific needs (e.g. guaranteed service provides delay guarantees) or for users with different needs (e.g. differentiated service). The introduction of complex value-added services raises the question of how we should define their "quality." While the QoS properties of current per-flow service classes are application independent and can be specified in fairly straightforward way, the quality of complex value-added services, such as video conferencing or distributed simulation, seems much harder to define and will in fact be application specific. Yet, service providers will want to differentiate themselves and users will want to request services of different quality, so it is important to define and support a notion of QoS that is applicable to these complex services.

In this paper we present a network architecture that explicitly supports the notion of application-oriented QoS for complex network services. Applications and service providers rely on *service brokers* to identify the network resources needed to satisfy application needs while meeting certain QoS and cost objectives. Brokers can be application-domain specific, i.e. they can incorporate a detailed understanding of an application domain allowing them to make intelligent tradeoffs and allowing them to interact with applications and service providers at a high level about QoS and optimization criteria. Service brokers can be hierarchical in the sense that one broker can invoke the services of another broker. Finally, service brokers can deal with heterogeneous networks and hierarchical resource management. A prototype of the proposed architecture has been implemented and we present some initial results.

The remainder of this paper is organized as follows. We first use a motivating example to identify key requirements of network support for application-oriented QoS. In Sections 3 through 6 we present our network architecture and elaborate on three key features: the specification of application-oriented QoS, support for resource and service hierarchies, and mapping onto heterogeneous network elements. In Section 7 we describe our prototype implementation and provide some examples. We present related work and summarize in Sections 8 and 9.

This work was sponsored by the Defense Advanced Research Projects Agency under contract N66001-96-C-8528.

2 Motivation

We sketch an example of a networking application whose implementation could benefit from application-oriented QoS features and discuss some of the requirements raised by this and similar applications.

2.1 Example

Let us consider a group of scientists who need connectivity for a video conference in which they will discuss the output of a running simulation. This is a many-to-many communication operation that includes bandwidth both to distribute the simulator output to the participants and to distribute video and voice from a small but changing subset of active participants. A service provider could be asked to deliver this service on a regular basis. The provider has a wide range of choices for how to implement this service: it can rely on bandwidth reservations for each of the flows, it can use best effort or differentiated service, or it can selectively pick a service class for each flow.

The issue of service quality comes up in a number of ways. An imprecise but important notion of QoS is the degree to which participants are satisfied with the service. This could for example be measured by having different service providers deliver the service and having participants rank the service providers. A more precise definition of service quality will come up in the contracting process: users may request video conferencing services of varying quality, and may use diverse criteria to evaluate quality; service providers will want to advertise a range of video services of different quality and cost. This more precise definition is likely to have both a per-flow requirement (e.g. minimum level of video quality) and more global requirements (e.g. relative value of flows or time to switch between video sources).

This example illustrates a number of properties of QoS for rich services. First, quality of service is application domain specific, e.g. criteria for the video conference will be very different from those for a service supporting nightly data backups. Second, different users will want to have different quality (and cost) levels, and service providers will want to differentiate themselves in terms of quality and cost. Finally, even for the same quality and cost level, different users may have different preferences, e.g. some users may attach more value to video quality than others.

Let us now assume that a service provider decides to deploy computation and storage resources in its network. These resources can, for example, be used to convert between video formats, or to provide aggressive compression to reduce the load on congested network links. These additional resources may allow the service provider to improve its quality/cost ratio, making it more competitive. Of course, providers can also deliver services that explicitly include computation and storage functions, e.g. a simulation or backup. Interestingly enough, broadening the types of resources and services has little impact on how we define service quality.

2.2 Design considerations

In the above example, we assumed that many-to-many connectivity requirements will be formulated as a single service request. The alternative is to issue multiple requests, one for each flow. Using a single request gives the service provider more opportunities for optimization: flow resources can be aggregated, multicast trees can be optimized, and computation resources can be allocated in the optimal locations. This means that for applications that have some structure and are planned, specifying the combined requirements at a high level is preferable. Note that other applications, e.g. traditional MBONE use, are more unstructured, so incremental allocation of resources on a per flow or per endpoint basis, as for example supported by RSVP, is more appropriate. Clearly, support is needed for a broad spectrum of application types.

Finally, we expect that network services will be structured hierarchically: complex services will be built on simpler ones. This is not unlike the structure of the traditional (non-electronic) services industry where general contractors often rely on the services of more specialized contractors. Quality of service plays a role in each of these interactions. A related issue is that of network heterogeneity. The underlying network that provides basic bit pipe services is not a single homogeneous entity, but consists of many service providers who are likely to offer a variety of service classes, e.g. different flavors of differentiated services. Service providers must be able to deal with this heterogeneity, i.e. it will be to their benefit if they can map application requirements onto some reasonable set of primitive service classes, and not just a single one.

3 Network Architecture

We sketch an architecture that provides the means to map an application's QoS needs onto a collection of specific resources. We also briefly describe Darwin, a prototype that implements some of the key components of this architecture. Subsequent sections elaborate on the key architectural components: the service broker, support for hierarchical services, and signaling support.

3.1 High-level architecture

The main players in our network are applications running on (collections of) endpoints, and network service providers offering services to applications and other providers. Applications that use the network can either locate and allocate resources themselves, or else contract with a service provider. Note that providing a service generally involves more than locating and allocating resources. Service providers possess proprietary expertise, e.g. software or protocols, that they use to provide unique functionality or to optimize cost or quality. Service providers can also rely on other service providers, either statically or dynamically, to provide part of their service functionality.

The process of allocating resources, either by an application or provider, has three components. The first component is resource discovery: locating available resources that can potentially be used to meet application requirements. This requires a resource discovery protocol. The second component is solving an optimization problem: identifying the resources that are needed to meet the application requirements, while optimizing quality and/or minimizing cost. Finally, the resources have to be allocated by contacting the providers that own them. In our architecture, the first two functions are performed by *service brokers*, while the third function is performed by a signaling protocol. Note that service brokers can be separate entities in the network, or they can be components of either an application or a service provider.

The main function of the service broker is to identify the resources that will satisfy a request in a way that optimizes a quality or cost metric. Since the quality metrics are highly application domain specific and since brokers have to make decisions and tradeoffs that will have a significant impact on the service quality, it follows that brokers need some expertise about the application domain. For example, when allocating bandwidth for a video stream, brokers have to understand the relative quality and cost (in terms of bandwidth) of different video encodings. We expect a wide variety of brokers with different expertise. Another aspect of having application domain expertise is that brokers can communicate with applications and providers in high-level domain-specific terms, e.g. a broker can receive a request for an MPEG-1 stream. Higher-level resource requests give the broker more flexibility in selecting resources.

An important feature of this approach is that service brokerage can be a recursive process. While simple service requests can typically be mapped onto specific resources, e.g. bandwidth allocations along a path in the network, more complex requests may involve the invocation of lower-level service requests. For example, while a request for a video transcoding service could be mapped onto a single-purpose transcoding node, it could equally well be satisfied by transcoding software plus a lower-level request for a general-purpose compute server. Thus service hierarchies can be constructed dynamically as well as statically.

In contrast to the service broker, the signaling protocol is application independent. The resources that have to be allocated are specified in application independent terms. However, the signaling protocol may have to deal with different types of networks, and if multiple signaling protocols emerge, this will be driven by differences in the network architectures, and not by application differences.

The brokers and signaling protocol need a common language for resource specification in order to communicate effectively. Service brokers will handle requests that include both generic resources (x Mbit/second), and domain-specific resource requests (MPEG-1 video stream). Different application communities will define the resource specifications that are relevant to them. Both resource discovery, and interactions between brokers and the signaling protocol will be based on resources that are expressed in a network-oriented way. Service specifications must be able to specify universally supported "standard" services, network-specific services (grade D1 premium), and complex services such as a request for an RSVP session. Note that to support value-added services in general, the language must include support for computing and storage resources.

3.2 Darwin

The CMU Darwin project is developing and implementing a set of resource management mechanisms that support the deployment of customizable, value-added services in the network [13]. Two key mechanisms that allow us to support application-oriented QoS are application-oriented service brokers and a matching signaling protocol. Two other mechanisms, the use of delegates to customize runtime traffic management and hierarchical scheduling, are outside the scope of this paper.

Figure 1 shows how the components in the Darwin system work together to manage network resources. Applications (1) running on end-points can submit requests for service (2) to a service broker (Xena). The service broker is implemented as a separate entity for flexibility; it could be part of an application or service provider in an actual deployment. The service broker identifies the resources needed to satisfy the request, and passes this information (3) to a signaling protocol, Beagle (4), which allocates the resources. For each resource, Beagle interacts with a local resource manager to acquire and set up the resource. The local resource manager modifies local state, such as that of the packet classifier and scheduler shown in the figure, so that the new application will receive the appropriate level of service.

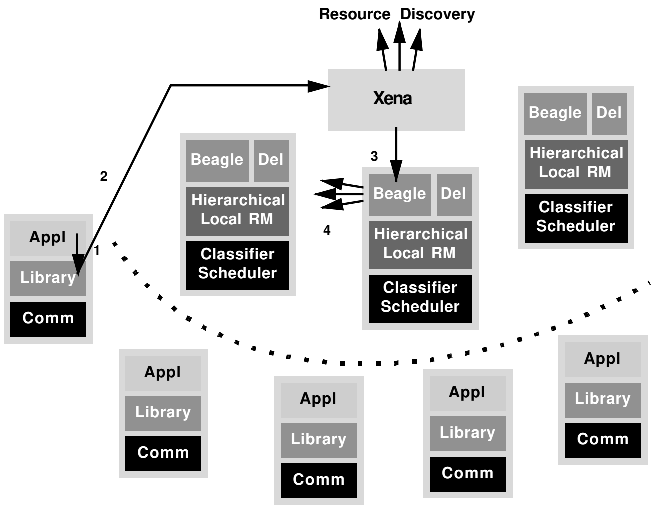

Figure 1: Darwin components

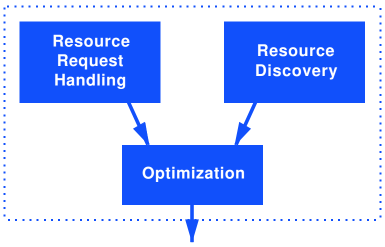

Figure 2: Broker components

The Darwin architecture is driven by our goal of supporting customizable services in a heterogeneous network environment. Service brokers interact directly with applications and can incorporate knowledge about the application domain, e.g. specialized knowledge about video conferencing. Local resource managers, on the other hand, concern themselves only with the management of a specific communication, computation or storage resource. We expect that different types of local resource managers will be deployed in different administrative domains. The signaling protocol has to bridge the gap: it receives high-level, application-independent and implementation-independent instructions from the service brokers, and translates them into specific instructions for each local resource manager. Thus it performs much the same complexity-reducing role of an intermediate representation in many other settings: it reduces the amount of code required to support m application domains on n network service types from $O(mn)$ to $O(m+n)$.

4 Application-oriented Service Brokers

The role of the service broker is to satisfy abstract resource allocation requests by selecting a collection of concrete resources, which then can be allocated through the use of a suitable signaling protocol. This selection is guided by optimization criteria specified by the application. Figure 2 shows the main functions of the service broker: interpreting the resource request, resource discovery, and optimized resource selection. We elaborate on each of these tasks, noting in particular the current capabilities of the Xena system.

4.1 Resource requests

At the highest level of the system, applications must request necessary resources through a brokerage interface. In the case of Xena, for example, an application describes its overall resource needs in the form of an *application input graph*. This is an annotated graph structure that specifies desired services (as nodes in the graph) and the communication flows that connect them (as edges). The annotations to the graph provide details of the resource allocation request; they vary in their level of abstraction from concrete specifications *(place this node at network address X)* to more abstract directives *(this node represents a service instance of class S)*. These annotations are directly related to the degree of control the application wishes to exert over the allocation: a mesh with fewer (or more abstract) constraints presents more opportunities for optimization than a highly specified one.

In the most constrained style of specification, the application specifies the network addresses where the services should be placed, the services themselves, and the quality of service desired of the flows that interconnect them. In this style of specification, the service broker's choices are limited to coarse routing: selecting communication service providers for the flows in the mesh. In a more relaxed specification, the application still specifies the service desired, but leaves its network address unspecified. This provides an additional degree of freedom: the broker now has the ability to place nodes *and* route flows.

In a more abstract form of specification, flow specifications no longer request a specific quality of service, but instead indicate the flow's semantic content. An example of a flow specification might be Motion JPEG with a specific frame rate and quality parameters. In addition to encapsulating low-level information that is needed to maintain meaningful application semantics, this approach offers several other advantages. Chief among these is the opportunity to perform *semantics-preserving* optimizations: mesh alterations that significantly modify flow data yet preserve its higher-level meaning. For example, when a Motion JPEG flow needs to cross a highly-congested network segment, there is the risk that the application will either fail or need to make do with a lower quality stream. However, using a matched pair of transcoders, a broker could convert the flow to the less communication-intensive (but more computation-intensive) MPEG format, and then recode it on the far side. In effect, the broker traded communication cost for computation cost, giving the application a higher quality of service than it could otherwise have attained in a potentially transparent way.

Another optimization that the semantic flow specifications enable is lowest-cost type unification; a group of nodes receiving the same multicast flow (say, a video stream) need to agree with the sender on the encoding that is used. If there is no single encoding acceptable to all parties, a broker can insert "type converter" nodes appropriately; according to its optimization criteria, it will strive to provide the highest quality solution that achieves type unification. The Xena prototype implements all of these capabilities.

4.2 Resource discovery

A key issue in the widespread deployment of the service broker model is the ability to advertise and discover service offerings. In general, the population of offerings will be dynamic in two ways: service providers will come and go, and their availability to provide a service will change as client sessions come and go. Scalability problems will make the global distribution of service information infeasible. Service discovery is the subject of ongoing attention in the networking and distributed computing communities. For example, the IETF has developed the Service Location Protocol (SLP)[15], which dynamically distributes service directory information over a campus-area network; wide-area extensions[12] are also being developed.

For cases in which brokers are asked to locate instances of specifically named services, SLP or a similar protocol can meet the need for service discovery. In order to support more complex matches, as required to exploit Xena's ability to make semantics-preserving transformations on data types, more powerful discovery mechanisms will be required in order to avoid an expensive combinatorial search of all possibilities. For example, if a broker wishes to establish a flow between nodes with matching semantics but mismatched encodings, it might wish to use two transducers that agree upon a common, but unspecified, format; those transducers cannot be found by a simple search. In practice, the size of the set of service offerings in the Internet will prevent exhaustive service discovery and evaluation for every request. Instead, brokers will cache the providers with which they have had recent successful interactions. The ability to cache information about service instances will, in general, greatly reduce the need for general-purpose resource discovery, much as knowing the phone number of a nearby pizzeria reduces trips to the yellow pages.

General resource discovery is outside of the scope of Xena. Instead, the current Xena implementation offers a mechanism through which services can register with a Xena instance. Xena then maintains a coarse representation of network topology (communication service providers and their interconnections), service layout (computation service providers and their locations), and an estimate of current resource utilization. This representation is by necessity only an approximation, both because network conditions are constantly changing and because service brokers are in general not privy to the internal state of service providers.

4.3 Resource selection and optimization

In general, a service broker should not only find a feasible solution to a request; it should attempt to find one that is optimal according to some application specified criteria. Using application specific criteria to guide resource selection is the key to support application-oriented QoS. The broker's job is to find a high-quality solution to a constrained optimization problem provided by the application. The constraints in the problem are criteria that must be met, e.g. entities at flow endpoints must agree with the type of data that will be exchanged. Given a set of candidate solutions that satisfy the constraints, the optimizer evaluates each according to its optimization criteria. In Xena, these optimization criteria are encoded by an application-specified objective function that maps candidate solutions to a numeric value: this function is composed of a sum of terms, where each term represents the "quality" of a particular layout choice. This allows applications to define service quality in an application-specific way. For tractability's sake the objective functions currently must be selected from a small set of built-in functions that represent useful objectives; e.g. minimize price, maximize throughput, minimize network delay.

Since problems of this nature are prohibitively expensive to solve exactly, a principal challenge in building the a resource optimizer is to develop an implementation that can provide high quality (but not, in general, optimal) solutions at a reasonable cost. Making the problem tractable will entail the use of heuristics and other simplifying assumptions that will necessarily trade quality for performance. The current implementation of Xena expresses its optimization problem in terms of an integer linear program, and turns it over to a solver package [2] that generates a sequence of successively better solutions at successively greater computation cost. We are currently evaluating the effectiveness of this approach.

5 Hierarchical Brokerage and Resource Management

Service brokers have to deal with two types of hierarchies. First, services are likely to be deployed in a hierarchical fashion, i.e. service brokers may invoke the help (services) of other services to satisfy parts of a request. Second, individual network resources may be shared and managed in a hierarchical fashion, i.e. a single physical resource is partitioned logically across multiple organizations, and these organizations can then further distribute the resource to multiple users. We elaborate on both the service broker and resource hierarchies, and discuss their impact on service broker implementation.

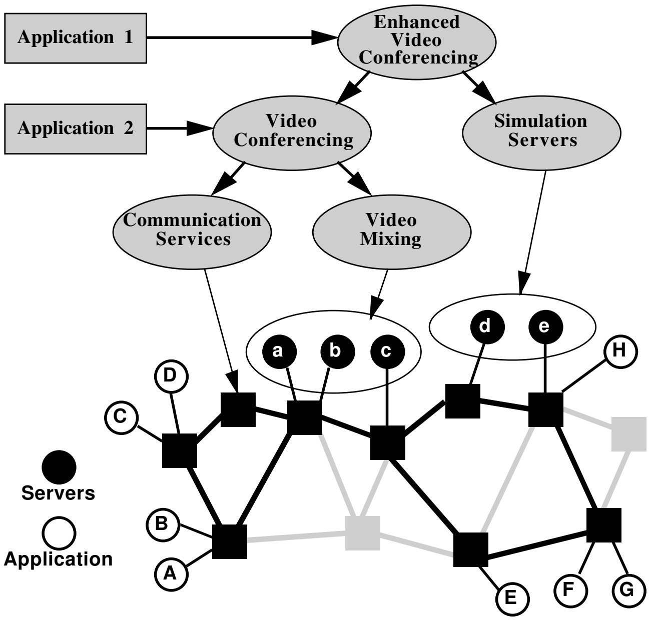

Figure 3: Hierarchy of service brokers

5.1 Service hierarchies

Figure 3 gives an example of a services hierarchy. An enhanced video conferencing service, like the example of Section 2, uses a broker that invokes the services of two other services: a simulation service that owns and operates compute nodes appropriate for simulation, and a video conferencing service. The video conferencing broker relies on two other services: a video mixing service that has both special-purpose and general-purpose hardware to manipulate real-time video streams, and a communication service provider. Since services rely on brokers to select resources and (invoked) services, every service hierarchy has a matching dynamic broker invocation hierarchy.

This example illustrates several features of service (and broker) hierarchies. First, the hierarchy has brokers for more complex services near the top, while more primitive brokers near the bottom satisfy requests from both "higher level" brokers

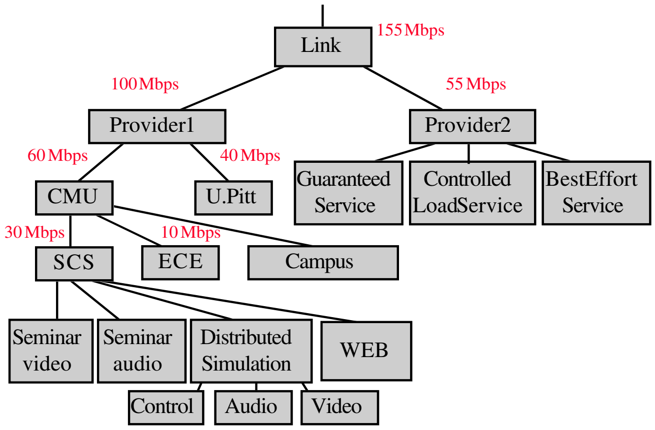

Figure 4: Resource tree for a link

and directly from applications. Not only the complexity of the services, but also the jobs of the brokers change as we move down in the hierarchy. The higher level brokers provide mostly ready-to-use services that feature significant application domain knowledge, considerable internal complexity, and great flexibility in how they can be implemented. In contrast, the brokers at the bottom of the hierarchy deal primarily in entities we think of more as resources than as services. They are similar, therefore, to the resource brokers that have been proposed in the network community [11].

The example also illustrates how applications and services are very similar and in some sense interchangeable. For example, an organization that has its own simulation engines can create an application that offer enhanced video conferencing by invoking the video conferencing service (Application 2 in Figure 3). Users will get the same service, although under the covers, some of the resources used are now internal to the organization.

5.2 Resource hierarchies

Figure 4 gives a simple example of a resource hierarchy. It shows how providers, organizations, and services share a specific resource, e.g. a network link or compute server. Each node in the tree controls a resource slice, plus it specifies a policy of how these resources should be shared by its children. "Control" can mean that the entity owns the resource, or that it leases it on a long-term or short-term basis. This type of resource sharing is sometimes referred to as link sharing [5, 14] in the networking community.

Resource hierarchies are fundamentally different from service broker hierarchies, even if the brokers primarily offer resources. A resource hierarchy describe how one specific resource is shared by organizations or applications. In contrast, a broker hierarchy describe how brokers rely on other brokers to provide services across the entire network. There is of course a relationship between the two: a node in a resource tree for a link identifies what fraction of that link a particular organization (e.g. a broker) can use. Thus a common scenario is that when a "producer" broker appears *below* a "consumer" broker in a service hierarchy, the producer appears *above* the consumer in the resource hierarchies for the physical resources used by the services for which they interact.

5.3 Impact on service broker implementation

The presence of both hierarchies has significant implications for the implementation of service brokers.

The first implication is that viewing brokers as "resource collectors" is overly simplistic. In fact, brokers can be very diverse. Some brokers are primarily "integrators": they operate by combining relatively high level services into even higher level ones. Others will primarily allocate resources, while some brokers will do a little bit of each. Different types of optimization of application-oriented QoS will be used by each of these brokers.

The second implication is that functionality of a service, and the resources that are used to provide the service, can be considered independently. The reason is that network links and switches, compute servers and storage servers can deliver a wide set of services, and, correspondingly, that any particular service can be provided by a broad set of resources. This separation of functionality and resources means that brokers can be used in a number of ways. One style of request can ask a broker to provide both a service and the resources needed for its implementation; this was the case for the example earlier in this section. Alternatively, an application or broker can ask a broker for resources and then implement its own functionality on top of it. Finally, an application or broker can request service from a broker, while specifying the resource "pool" that should be used. An example of this would be an application at CMU asking a broker to set up a video conference, while specifying that it should use a specific provider of communication resources with which CMU has a long-term or preferred relationship. Service providers have similar options: they can use internal resources or rely on resource brokers.

Finally, in an environment where different organizations often contribute service functionality and resources, resource authorization is a critical issue. Resource brokers have to know what resource pools they are allowed to use for each request, and for third party resources, they must be able to verify that the originator of the request is allowed to use them. The issue of resource authorization is not limited to brokers. For example, the signaling protocol and local resource managers must be able to validate the resource allocation request and set up appropriate charging.

6 Signaling

Once a broker has identified the resources needed to satisfy a request, it needs a signaling protocol to allocate the resources. In contrast to traditional signaling protocols such as RSVP [18] and PNNI [1], which operate on the basis of individual flows, this signaling protocol will be asked to allocate communication resources for a complete set of flows, plus possibly computation and storage resources. The most obvious advantage of such an application-oriented signaling protocol is efficiency: since all information if known up front, resource allocation can be done more efficiently and faster. For example, if multiple application flows use the same link, which will be common, the appropriate state can be installed using a single message that specifies (1) what resources the application needs and (2) (optional) how they should be distributed over multiple flows. If hierarchical resource management is used, the effect of the message would be that the local resource manager of the link would add a node or small subtree to the resource hierarchy for the link.

The signaling protocol has to perform a number of other tasks related to our goal of providing application-oriented QoS over

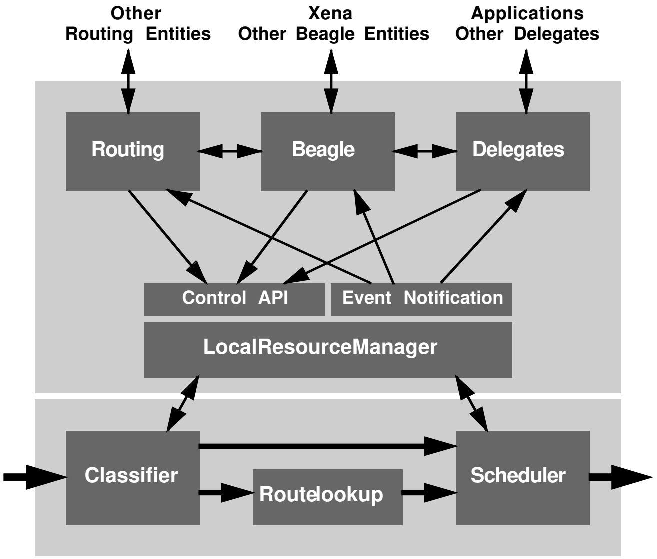

Figure 5: Darwin node software architecture

heterogeneous networks. For example, while some per-flow QoS and resource specifications can simply be passed on to the local resource management, others may have to be remapped. For example, "guaranteed bandwidth" per-flow QoS may be specified in slightly different ways in different networks, so the signaling protocol may have to translate between parameter formats. Note also that the hierarchical nature of service providers can result in hierarchical signaling invocations. For example, a signaling protocol may have to call another signaling protocol (e.g. RSVP) to allocated resources for a service (e.g. an RSVP session) in a specific subnet.

The Darwin signaling protocol is called Beagle, and a simple version is operational. While there is room for optimization and refinement, it can allocate communication resources and install delegates. It also knows how to deal with resource hierarchies and can optimize resource allocation through resource sharing, if given information about synchronization patterns between application traffic sources. The current implementation of Beagle is based on RSVP [19].

7 Darwin

We describe the Darwin prototype and illustrate Xena using a simple example.

7.1 Darwin prototype implementation

The Darwin prototype runs on a network testbed consisting of three routers and about ten endpoints. The routers are PCs running NetBSD, while the diverse set of hosts (DEC Alpha running DEC Unix, PCs running FreeBSD or NetBSD) can run applications or function as load generators.

Figure 5 shows the software architecture of the routers. The bottom part of the picture corresponds to the data plane. The focus in this component is on simplicity and high throughput, and the overall structure of this component is similar to what is for example proposed in the RSVP and IntServ IETF working groups. The top half corresponds to the control plane. Activities in the control plane happen on a coarser time scale, and, although there is only a limited set of resources available to support control activities, there is more room for customization and intelligent decision making. The figure illustrates how Beagle

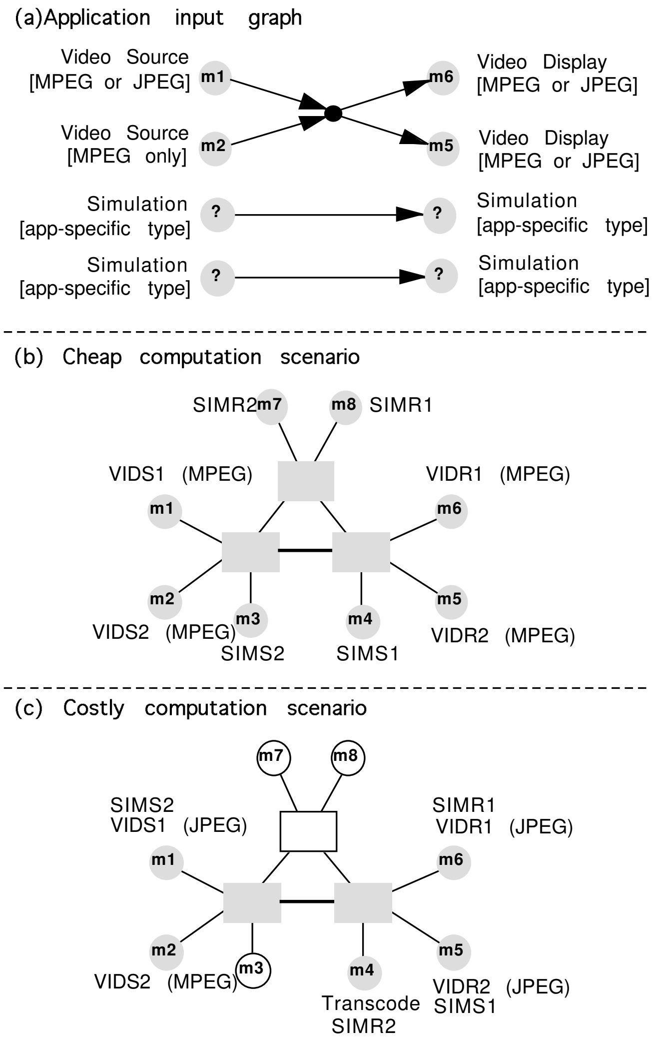

Figure 6: Resource optimizations under two cost scenarios

takes a specification for resources, interprets and translates it into a format that is appropriate for the local resource manager, and finally issues a request to the local resource manager through the resource API. Beagle can also set up "delegates" that can support customized resource management at runtime [13].

Xena is implemented as a separate process and can run anywhere in the network.

7.2 Xena Example

We imagine four scientists collaborating on a distributed simulation; the scientists have shared access to an eight-node distributed computing testbed (depicted in Figure 6(b)), that also supports other applications. The scientists communicate with one another using the videoconference facility, meanwhile controlling the distributed computation.

Figure 6(a) shows the application input graph supplied to Xena. For the sake of clarity, some detail has been omitted; for example, we have depicted communication flowing in one direction only. The application has provided specific network addresses (`m1`, `m2`, `m5`, `m6`) for the nodes participating in the videoconference because our chatty scientists are physically located at those machines. The nodes are also annotated by service type desired (Video Source, Video Display); since the services are already registered with Xena, it is already aware of the type configurations of which they are capable. In particular, note that the video source at `m2` is capable of emitting only MPEG. The nodes are connected by a multipoint-to-multipoint flow (only partially depicted). This flow specification describes only the connectivity between the nodes; the precise type (encoding) of the video streams and their Quality of Service will be derived by Xena from the rest of the specification.

The application did not specify what nodes should participate in the distributed simulation, so these addresses will have to be supplied by Xena's placement algorithm. For the data flows, Xena has to only find a mutually compatible type configuration and perform coarse routing. Because the flows have a private application-defined type, i.e. a type that Xena does not have any information on, Xena cannot perform any optimizations involving filter insertion for this part of the application. The application request also specifies that a minimum level of video quality must be achieved and that cost must be minimized.

In Figure 6(b) and (c) we depict two different resource "meshes" generated by Xena; the shaded elements are used for the scientists' application, while the white elements are not. These meshes were generated by applying the Xena algorithm to the above problem under two different load scenarios. In the lightly loaded scenario, we assume plentiful computation resources but scarce communication resources; hence the communication resources will be expensive and Xena should minimize their use, trading computation for communication wherever possible. In the heavily loaded scenario, we have the opposite situation; expensive computation should shift the balance toward communication. We will refer to these two scenarios as the cheap and costly computation scenario, respectively.

In order to simulate the above load scenarios, we present the following price schedules to Xena. While the values used in this example are somewhat arbitrary, they nevertheless capture the relationships between computation and communication that are present in a real system. In the lightly loaded scenario, computation is free. Communication is charged on a per-connection basis, according to the amount of bandwidth used. Specifically, a stream carrying Motion JPEG costs 2 units, one carrying MPEG costs 1, and one carrying simulation data costs 5.

In the heavily loaded scenario, the communication costs are the same but we now charge for computation: 1 unit for JPEG compression/decompression, 4 units for MPEG, and 5 units for the computation needs of each simulation node. Additionally, there is a charge of 2 units if one of the hardware-based transcoders, available at m3 and m4, are used. Finally, we impose a heavy penalty (10 units) for the use of m3, m7, and m8 which is meant to reflect that they are already in use by an unrelated application.

In both scenarios we impose a 1 unit overlap penalty when two services are placed at the same computation node. This is for the purposes of load balancing, and will tend to spread services evenly among servers, all other things being equal.

The results in Figure 6 show how in the cheap computation scenario, Xena distributes the computational load, thus avoiding the overlap penalty for computations. For the video streams it uses the lower bandwidth MPEG video encoding to minimize the use of expensive communication resources, even though it is more expensive to encode and decode the MPEG. In contrast, in the costly computation scenario, Xena avoids the use of the already heavily loaded servers (m3, m7, m8), and instead places the simulation tasks primarily on the scientist's machines.

Moreover, video flows are JPEG encoded, which requires more bandwidth (which is plentiful), but is computationally less expensive. To accommodate m2, which supports only MPEG, it transparently inserts a video transcoder.

8 Related work

The QoS architecture presented here is in many respects an evolution of traditional structures. For example, the resource management mechanisms for the Internet that have been defined in the IETF in the last few years rely on QoS routing [17, 9, 6] (resource brokers), RSVP [18] (signaling), and local resource managers that set up packet classifiers and schedulers. The more recent proposals for differentiated service [11] require similar entities. The specific responsibilities of the entities differ, of course, in these proposals. In Darwin, we emphasize the need for hierarchical service invocation, hierarchical resource management (link sharing), customization of resource management, and support for not only communication, but also computation and storage resources.

Several recent and ongoing projects touch on intersecting issues. The Telecommunications Information Networking Architecture (TINA) [4] includes a service architecture that provides a uniform platform for services. This architecture supports a service hierarchy, but does not have any notion of resource hierarchies. Since the primary focus in TINA is mechanisms that enable the rapid creation and deployment of value-added services, it has a limited notion of customization (or application-oriented QoS), but, unlike Darwin, it does not address optimization of resource allocation using application-domain knowledge. Globus, a system that supports distributed supercomputing, includes a resource management architecture [3] based on resource brokers, and incorporating a notion of co-allocation that could be used to address the question of global optimization of resource selection. However, Globus does not directly take communication or quality of service into account, and does not address the issue of service or resource hierarchies. Nahrstedt and Smith [10] describe the concept of a QoS broker, which incorporates the more traditional aspects of resource broker functionality. Like Globus, their architecture does not address service or resource hierarchies, or invocation-time binding of brokers to resources.

Other projects are concerned with specific technology that could be suitable for constructing a service broker architecture. X-Bind [7] is a middleware toolkit for building scalable multimedia services on top of heterogeneous computing and networking platforms. It provides a CORBA based programming environment for dynamic creation and composition of services. X-Bind is primarily a framework on top of which Darwin style resource and service brokers could be built. In contrast to Darwin, X-Bind itself does not specify any algorithms for providing application-oriented QoS. QuO [8] is a system that provides tools for specifying and enforcing quality of service contracts between service providers and clients. While it has no built-in notion of service hierarchy, its capabilities could be used in a broker-based system. In a more formal vein, the Quasar project [16] is developing tools for building application-oriented QoS languages and translating them into lower-level specifications. This approach too could potentially be used in a service broker architecture.

9 Conclusion

We have described a network service architecture that provides for the specification and provisioning of application-oriented quality of service measures through the use of *service brokers*. These brokers encapsulate application-specific QoS concerns, optimize resource selection based on application objectives, and form the basis for dynamically constructed hierarchies of value-added services. The mechanisms in this paper, most of which have already been implemented in the Darwin prototype, promise to improve both the ease of creation and efficiency of invocation of ambitious new network applications.

Acknowledgement

We would like to thank our colleagues in the Darwin project, specifically Eugene Ng, Ion Stoica, Eduardo Takahashi, and Hui Zhang, for fruitful discussions on service hierarchies and for their contributions to the Darwin design and implementation.

References

[1] Private Network-Network Interface Specification Version 1.0, March 1996. ATM Forum document - af-pnni-0055.000.

[2] Michel Berkelaar. lp_solve: a Mixed Integer Linear Program solver. ftp://ftp.es.ele.tue.nl/pub/lp_solve/, September 1997.

[3] K. Czajkowski, I. Foster, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. Draft at http://www.globus.org/globus/papers.htm, 1998.

[4] F. Dupuy, C. Nilsson, and Y. Inoue. The TINA Consortium: Toward Networking Telecommunications Information Services. *IEEE Communications Magazine*, 33(11):78–83, November 1995.

[5] Sally Floyd and Van Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, August 1995.

[6] R. Guerin, A. Orda, and D. Williams. QoS Routing Mechanisms and OSPF Extensions. *IETF Internet Draft <draft-guerin-qos-routing-ospf-00.txt>*, November 1996.

[7] A. Lazar, Koon-Seng Lim, and F. Marconcini. Realizing a foundation for programmability of atm networks with the binding architecture. *IEEE Journal on Selected Areas in Communication*, 14(7):1214–1227, September 1996.

[8] Joseph P. Loyall, Richard E. Schantz, John A. Zinky, and David E. Bakken. Specifying and measuring quality of service in distributed object systems. In *Proceedings of ISORC '98*. IEEE, April 1998.

[9] Qingming Ma and Peter Steenkiste. On path selection for traffic with bandwidth guarantees. In *Fifth IEEE International Conference on Network Protocols*, pages –, Atlanta, October 1997. IEEE.

[10] Klara Nahrstedt and Jonathan M. Smith. The QoS Broker. *IEEE Multimedia*, 2(1):53–67, Spring 1995.

[11] K. Nichols, V. Jacobson, and L. Zhang. A Two-bit Differentiated Services Architecture for the Internet, Nov 1997. Internet draft, draft-nichols-diff-svc-arch-00.txt, work in progress.

[12] J. Rosenberg, H. Schulzrinne, and B. Suter. Wide Area Network Service Location. work in progress, Internet Draft, draft-ietf-srvloc-wasrv-01.txt, November 1997.

[13] Peter Steenkiste, Allan Fisher, and Hui Zhang. Darwin: Resource management in application-aware networks. Technical Report CMU-CS-97-195, Carnegie Mellon University, December 1997.

[14] Ion Stoica, Hui Zhang, and T. S. Eugene Ng. A Hierarchical Fair Service Curve Algorithm for Link-Sharing, Real-Time and Priority Service. In *Proceedings of the SIGCOMM '97 Symposium on Communications Architectures and Protocols*, pages 249–262, Cannes, September 1997. ACM.

[15] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. Service Location Protocol. Request for Comments 2165, June 1997.

[16] Jonathan Walpole. The Quasar Project. http://www.cse.ogi.edu/DISC/projects/quasar/.

[17] Z. Wang and J. Crowcroft. Quality-of-Service Routing for Supporting Multimedia Applications. *IEEE JSAC*, 14(7):1288–1234, September 1996.

[18] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. Rsvp: A new resource reservation protocol. *IEEE Communications Magazine*, 31(9):8–18, September 1993.

[19] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource Reservation Protocol. *IEEE Communications Magazine*, 31(9):8–18, September 1993.