Darwin: Customizable Resource Management for Value-Added Network Services

Prashant Chandra, Yang-hua Chu, Allan Fisher, Jun Gao, Corey Kosak, T. S. Eugene Ng, Peter Steenkiste, Eduardo Takahashi, and Hui Zhang, Carnegie Mellon University

Abstract

The Internet is rapidly changing from a set of wires and switches that carry packets into a sophisticated infrastructure that delivers a set of complex value-added services to end users. Services can range from bit transport all the way up to distributed value-added services like video teleconferencing, virtual private networking, data mining, and distributed interactive simulations. Before such services can be supported in a general and dynamic manner, we have to develop appropriate resource management mechanisms. These resource management mechanisms must make it possible to identify and allocate resources that meet service or application requirements, support both isolation and controlled dynamic sharing of resources across services and applications sharing physical resources, and be customizable so services and applications can tailor resource usage to optimize their performance. The Darwin project has developed a set of customizable resource management mechanisms that support value-added services. In this article we present and motivate these mechanisms, describe their implementation in a prototype system, and describe the results of a series of proof-of-concept experiments.

There is a flurry of activity in the networking community developing advanced services networks. Although the focus of these efforts varies widely from per-flow service definitions like integrated services (IntServ) [1, 2] to service frameworks like Xbind [3], they share the overall goal of evolving the Internet service model from what is essentially a basic bitway pipe to a sophisticated infrastructure capable of supporting novel advanced services.

In this article we consider a network environment that comprises not only communication services, but storage and computation resources as well. By packaging storage/computation resources together with communication services, value-added service providers will be able to support sophisticated services such as intelligent caching, video/audio transcoding and mixing, virtual private networking, virtual reality games, and data mining. In such a service-oriented network, value-added services can be composed in a hierarchical fashion: applications invoke high-level service providers, which may in turn invoke services from lower-level service providers. Providers in the top of the hierarchy will typically integrate and add value to lower-level services, while the lowest-level services will supply basic communication and computational support. Since services can be composed hierarchically, both applications and service providers will be able to combine their own resources with resources or services delivered by other service providers to create a high-quality service for their clients. The design of such a service-oriented network poses challenges in several areas, such as resource discovery, resource management, service composition, billing, and security. In this article we focus on the resource management architecture and algorithms for such a network.

Service-oriented networks have several important differences from traditional networks that make existing network resource management inadequate. First, while traditional communication-oriented network services are provided by switches and links, value-added services will have to manage a broader set of resources that includes computation, storage, and services from other providers. Moreover, interdependencies between differ-

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

0890-8044/01/$10.00 © 2001 IEEE

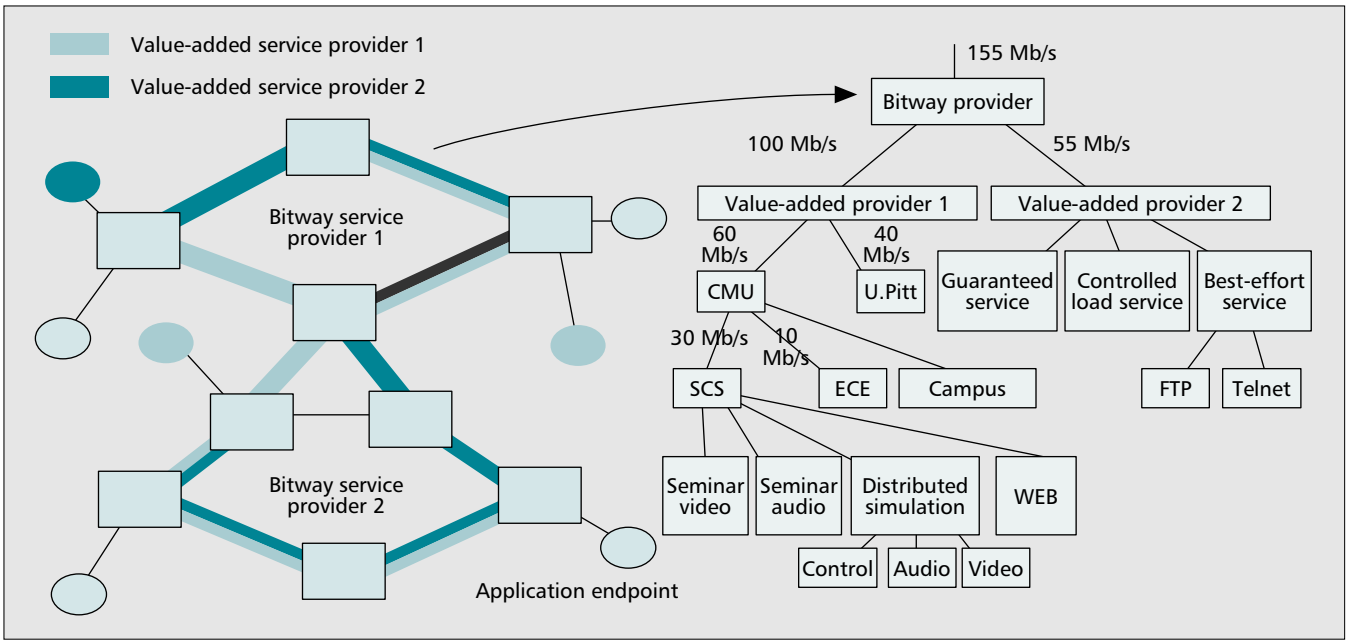


Figure 1. *An example resource management hierarchy.*

ent types of resources can be exploited by value-added service providers to achieve higher efficiency. For example, using compression techniques, one can make trade-offs between network bandwidth and CPU cycles. Furthermore, value-added services are likely to have service-specific notions of quality of service (QoS) that would be difficult to capture in any fixed framework provided by the network. Therefore, the network must allow service providers to make resource trade-offs based on their own notion of QoS. Finally, the resources allocated to applications or providers will often not be reserved for their exclusive use, but be shared dynamically with other applications and providers for efficiency reasons. As a result, resource availability will continuously evolve, and providers will have to be able to make the above trade-offs not only at startup, but also on an ongoing basis. The challenge is to accommodate service-specific qualities of service and resource management policies for a large number of service providers.

To support these new requirements, we argue that resource management mechanisms should be flexible so that resource management policies are *customizable* by applications and service providers. We identify three dimensions along which this customization is needed: space (which network resources are needed), time (how the resources are applied over time), and services (how the resources are shared among different service providers and applications). Additionally, the mechanisms along all three dimensions need to be integrated so that they can leverage off one another.

The Darwin project has developed an integrated set of customizable resource management tools that support value-added electronic services. In this article we first outline and motivate the Darwin architecture. We describe each of the four components of the Darwin architecture in more detail, and illustrate their operation using several proof-of-concept experiments. We demonstrate the integrated operation of Darwin, discuss related work, and then summarize.

Integrated Customizable Resource Management

We discuss the requirements that complex value-added electronic services impose on the network infrastructure. We propose the virtual mesh as a core abstraction that captures these requirements and derive the resource management support needed to realize the virtual mesh abstraction. The next section describes how Darwin addresses these resource management requirements.

Service Requirements

Our focus is on complex value-added electronic services. *Complex* means that the services involve multiple endpoints and use a wide variety of data types with challenging timing and synchronization constraints. *Value-added* means that the services integrate low-level communication, computation, and storage services, and provide high-level functionalities. Examples include distributed simulation, interactive games, and telepresence. For complex services, the definition of service quality is service-specific, and hardwired QoS mechanisms will not be able to adequately support such a diverse set of services. Instead, we need a set of resource management mechanisms that are customizable. Service providers must be able to influence how their notion of QoS is mapped onto low-level QoS mechanisms (e.g., IntServ or DiffServ), and to tailor the selection and use of resources to fit their specific needs.

We also expect electronic services to be structured in a hierarchical fashion: value-added services are implemented in terms of lower-level services. The motivation for this is reuse of functionality and the need for specialization; for example, providers will build on specialized service providers rather than duplicating effort and expertise in those areas. Hierarchical services require support for hierarchical resource management. While there has been some work in this area (e.g., research on link-sharing [4]), we need mechanisms that support a wide range of sharing policies and large dynamic hierarchies [5, 6].

Virtual Meshes

In Darwin we use a virtual mesh, sometimes also called a virtual network or supranet [7], as a core abstraction for resource management in value-added service networks. A virtual mesh is the set of resources allocated and managed in an integrated fashion to meet the needs of service providers or applications. A virtual mesh does not contain physical resources, but rather slices of physical resources that are shared with other providers. A key feature of virtual meshes is that providers

can manage and control the resources in "their" virtual mesh in a way that best fits their needs (i.e., customized management). For example, they can specify which of their customers may use virtual mesh resources and impose sharing policies for those customers, similar to what they could do with physical resources. A virtual mesh should not be confused with a virtual private network (VPN), which is a pure networking concept. A VPN only contains link and router resources, while meshes typically also have other resources, and people expect VPNs to support specific protocols (e.g., for routing and network management) that may not be present in all virtual meshes.

Combining the notion of a virtual mesh with that of a service hierarchy results in a hierarchy of virtual meshes. Focusing on a specific resource (e.g., a link) reveals a resource tree that captures how that resource is shared. Figure 1 shows the virtual meshes of two value-added service providers and the resource tree for one of the links they share. The root node represents the physical link resource; the leaf nodes represent the finest granularity of virtual resources, which can be individual flows or flow aggregates (e.g., a best-effort traffic class). Interior nodes represent virtual resources managed by entities such as service providers, organizations, and applications. The set of edges connecting a parent to its children specify how parent resources are shared by the children.

Resource Management Requirements

There are several requirements for managing a hierarchy of virtual meshes to support value-added services. These requirements can be organized into the following classes:

- *Resource type and location*: Services need to be able to allocate and manage a rich set of resources (links, switch capacity, storage capacity, compute resources, etc.) in virtual meshes to meet their needs. Resources should be allocated in a coordinated way, since there are often dependencies between resources, and there should be support for global optimization of resource use.
- *Resource sharing across services*: Service providers will want to reserve some resources so they can meet certain minimal QoS requirements; at the same time, they will want to dynamically share resources for efficiency. Mechanisms are needed to isolate or dynamically share resources in a controlled fashion across service providers and applications.
- *Resource availability over time*: Conditions in the network and user requirements change over time. Services must be able to quickly change how resources are used and allocated, so they can operate under a broad range of conditions.

Because of the broad diversity in services and since service providers will want to differentiate themselves from their competitors, there is a strong need for customization that cuts across these three dimensions: providers will want to customize which resources they allocate, how they share resources with other service providers, and how they adapt their resource use over time.

Darwin

The Darwin system comprises a set of integrated resource management mechanisms that support value-added electronic services. In this section we present the mechanisms and discuss how they work together.

Darwin Resource Management Mechanisms

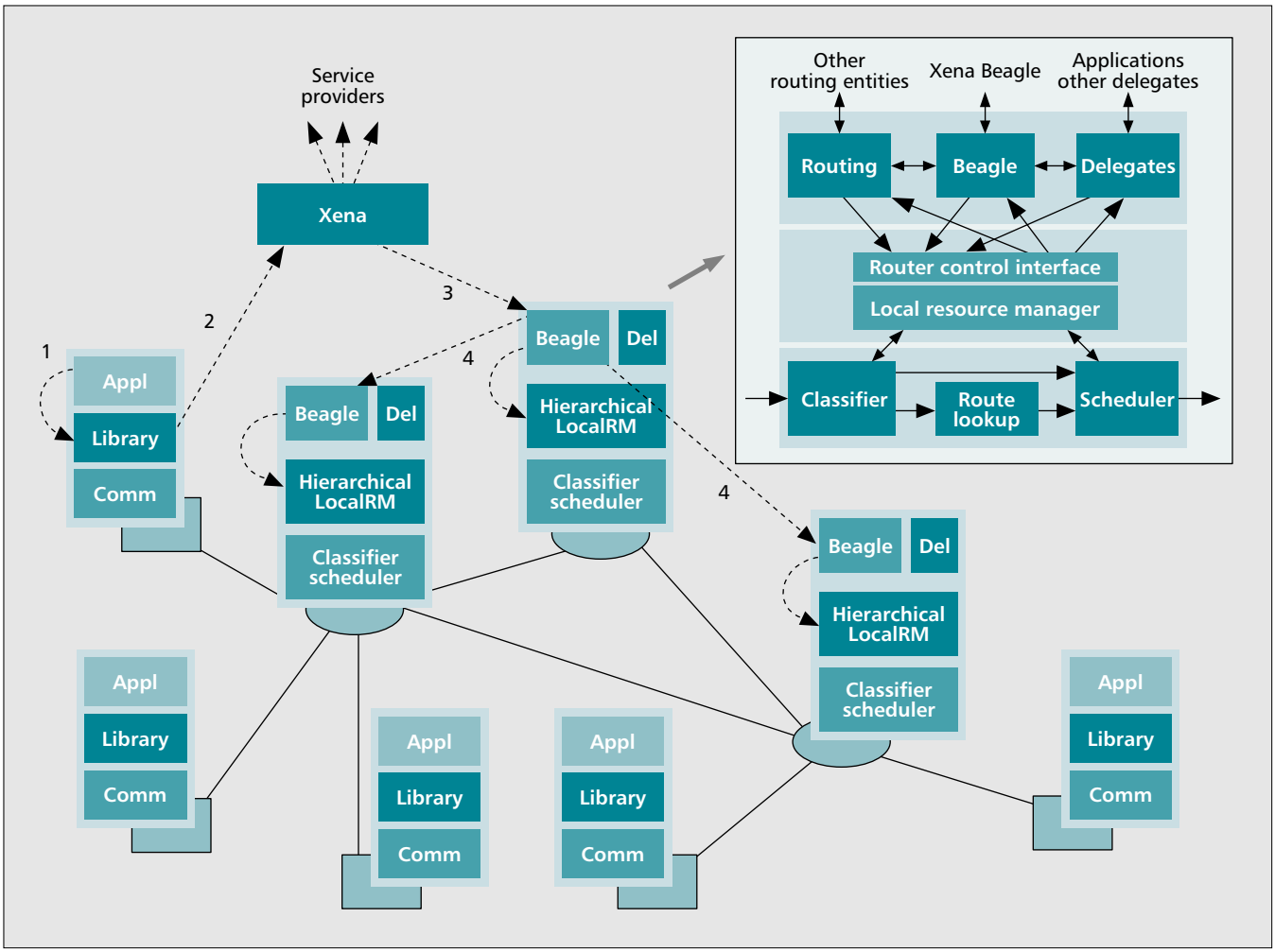
The Darwin architecture consists of four resource management mechanisms:

- **High-level resource selection**: This mechanism, sometimes called a resource or service broker, performs global selection of the resources based on a high-level application request, typically using domain knowledge for optimization. Its tasks include performing trade-offs between services (e.g., trading computation for communication) according to the application-selected value metric (e.g., maximizing QoS). It must also perform coordinated allocations for interdependent resources (e.g., since the amount of processor power required by a software transcoder is proportional to the bandwidth of video data flowing through it, these two allocations must be correlated.) We must also be able to interconnect incompatible services or endpoints, for example, by automatically inserting a transcoder service between two otherwise incompatible videoconference participants. The Darwin system contains an example broker called *Xena*.
- **Runtime resource management**: This mechanism injects application- or service-specific dynamic behavior into the network. Rather than performing runtime adaptation at flow endpoints (where the information provided by network feedback is potentially stale and inaccurate), it allows rapid local runtime adaptation at the switching points in the network's interior in response to changes in network behavior. Darwin runtime customization is based on *control delegates*, Java code segments that execute on routers.
- **Hierarchical scheduling**: This mechanism provides isolation and controlled sharing of individual resources among service providers. For each physical resource, sharing and thus contention exist at multiple levels: at the physical resource level among multiple service providers, at the service provider level among lower-level service providers, and at the application level among individual flows. The hierarchical scheduler allows various entities (resource owners, service providers, applications) to independently specify different resource sharing policies and ensure that all these policy requirements are satisfied simultaneously. Darwin uses the Hierarchical Fair Service Curve (H-FSC) scheduler.
- **Low-level resource allocation**: This mechanism is a resource allocation protocol that provides an interface between Xena's abstract view of the network and low-level network resources. It has to allocate real network resources (bandwidth, buffers, cycles, memory) while hiding the details of network heterogeneity from Xena. The Darwin resource allocation protocol is called *Beagle*.

These mechanisms were chosen because they cover the type/location, sharing, and time requirements outlined above. As shown in Table 1, hierarchical

Mechanism	Operations		
	Time scale	Scope of information and actions	Customization
Xena	Coarse	Global, high-level	Domain-specific optimizations
Delegates	Medium	Restricted, detailed	Customized policies and actions
Hierarchical scheduler	Fine	Local, network-specific	Network parameters
Beagle	Coordinates all other mechanisms		

Table 1. *Complementary nature of the Darwin mechanisms.*



■ Figure 2. *Darwin node software architecture.*

scheduling, delegates, and Xena handle resource management on different time scales (packet forwarding times, round-trip times, and seconds, respectively). The difference in time scale impacts the complexity and scope of the decisions that can be made by the mechanisms. Xena can perform sophisticated resource optimization using global information on network status and application requirements. On the other hand, the scheduler can make only simple decisions using local information. The flexibility of customization differs similarly across the three mechanisms. The Beagle resource allocation protocol coordinates the activities of the mechanisms so they can operate in an integrated fashion. Beagle is also responsible for translating the high-level resource allocations into network-specific requests.

The four Darwin mechanisms correspond directly to simple operations on a virtual mesh. Xena identifies the resources that make up a virtual mesh, while Beagle allocates the virtual mesh resources by contacting individual resource managers. The hierarchical scheduler enforces the boundaries between different virtual meshes, while delegates do runtime management for the resources inside the virtual mesh with which they are associated.

Darwin System Architecture

Figure 2 shows how the components in the Darwin system work together to manage network resources. Applications (1) running on endpoints can submit requests for service (2) to a resource broker (Xena). The resource 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. The signaling protocol can also set up delegates. Throughout this process, appropriate resource authorizations must be made. Resource brokers have to know which resource pools they are allowed to use, and the signaling protocol and local resource managers must be able to validate the resource allocation request and set up appropriate billing or charging.

Figure 2 also shows a more detailed view of the router architecture. The bottom layer of the architecture corresponds to the data plane. The focus in this component is on simplicity and high throughput. The top layer corresponds to the control plane. Activities in the control plane happen on a coarser time scale; although there is only a limited set of resources available to support control activities, there is more room in the control plane for customization and intelligent decision making. The local resource manager will in general execute on a CPU close to the data path. Routing, signaling, and delegates, on the other hand, are not as tightly coupled to the data path, and could run on a separate processor.

The Darwin architecture is similar in many ways to traditional resource management structures. For example, the resource management mechanisms for the Internet that have been defined in the Internet Engineering Task Force (IETF)

in the last few years rely on QoS routing (resource brokers), Resource Reservation Protocol (RSVP) [8] (signaling similar to Beagle), and local resource managers that set up packet classifiers and schedulers. The more recent proposals for DiffServ [9] require similar entities. The specific responsibilities of the entities differ, of course, in these proposals. In Darwin we emphasize the need for customization of resource management, hierarchical resource management (link sharing), and support for resources including not only communication, but also computation and storage.

The Darwin system has been implemented for PC-based routers running the NetBSD and FreeBSD operating systems. The code for the Darwin scheduler, the delegate runtime environment, and the Beagle resource allocation protocol has been released for use by the research community and is available from our Web site http://www.cs.cmu.edu/~darwin.

Darwin Usage

The Darwin resource management mechanisms can be used in a variety of ways, but uses can roughly be classified into two classes. The first class consists of applications that have been modified to explicitly use some or all of the Darwin mechanisms. For example, a videoconferencing application could explicitly call a resource broker to select the set of resources (e.g., transcoders, network bandwidth) that will optimize the quality of the video conferencing session, and use Beagle to allocate those resources and install delegates as needed. We present an example of such an application later. Alternatively, the Darwin mechanisms can be used by network services that support end users indirectly; that is, end users and applications running on endpoints benefit from the Darwin mechanisms without using Darwin directly. For example, Darwin can be used to support rich VPN services [10] that allow network managers to provide QoS guarantees for certain classes of traffic without requiring modifications to the applications.

The Darwin mechanisms were designed to work together effectively, and throughout this article we will assume that the Darwin mechanisms are used in an integrated fashion. However, each mechanism can also be used in isolation. For example, Beagle can be used to allocate network resources without necessarily using Xena to select the resources, or for networks that use schedulers other than the H-FSC scheduler. Similarly, the H-FSC scheduler can be used to support bandwidth reservations without needing the other Darwin mechanisms.

The next four sections describe Xena, delegates, hierarchical scheduling, and Beagle in more detail.

Xena: Selection of Resource Type and Location

The process of allocating resources, by either an application or a provider, has three components. The first is resource discovery: locating available resources that can potentially be used to meet application requirements. The second is solving an optimization problem: identifying the resources needed to meet the application requirements, while maximizing 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 a service broker called Xena, while the third function is performed by Beagle.

Xena Design

The application submits its resource request to Xena in the form of an *application input graph*, an annotated graph structure that specifies desired services (as nodes in the graph) and the communication flows that connect them (as edges). The annotations can vary in their level of abstraction from concrete specifications (*place this node at network address X*) to more abstract directives (*this node requires a service 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 specification, the application specifies the network addresses where the services should be placed, the services themselves, and the QoS parameters for the flows that connect them. In this style of specification, Xena's optimization opportunities are limited to coarse routing: selecting communication service providers for the flows in the mesh. In a less constrained specification, the application can leave the network address of a service unspecified. This provides an additional degree of freedom: Xena now has the ability to place nodes *and* route flows. In addition, the application can leave the exact QoS parameters unspecified, but instead indicate the flow's semantic content. An example of a flow specification might be Motion JPEG, with specific frame rate and quality parameters. In addition to providing sufficient information to maintain meaningful application semantics, this approach gives Xena the opportunity to optimize cost or quality by inserting *semantics-preserving* transformations to the mesh. For example, when a Motion JPEG flow needs to cross a congested network segment, Xena can insert a matched pair of transcoders at two ends of the network segment. The first transcoder converts the flow to a more bandwidth-efficient coding format (e.g., MPEG or H.261) and then convert it back to JPEG on the far side. Another optimization is the 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 used. If there is no single encoding acceptable to all parties, Xena can insert "type converter" nodes appropriately.

A feasible solution to the resource selection problem is one that satisfies all the constraints based on service-specific knowledge or application specification; for example, entities at flow endpoints must agree with the type of data to be exchanged. Given a set of feasible solutions, Xena evaluates each according to the 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 QoS in an application-specific way. By using application-specific criteria to guide resource selection, Xena in effect allows applications to customize the definition of QoS.

Xena Implementation

Our current implementation of Xena includes the interfaces to the other system entities (applications, service providers, and Beagle), plus the solving engine and optimizations described above. The application interface allows the specification of requests that include nodes, flows, types, and transcoders. The current Xena implementation does not have a general resource discovery protocol. Instead, it offers a mechanism through which services can register their availability and capabilities (i.e., a simple publish-subscribe mechanism). This information allows Xena to build a coarse database of available communication and computation resources, and an estimate of their current utilization. Additionally, Xena maintains a database that maps service and flow types (e.g., transcoding or transmitting MPEG of a certain quality) to their effective resource requirements (e.g., CPU cycles or megabits per second). Finally, there is a database that contains the various semantics-preserving transformations (e.g.,

JPEG-to-MPEG) and how to instantiate them.

The resource optimization problem can be addressed in many ways. In the first Xena implementation we express it as a 0-1 integer programming problem and hand it to a solver package [11] that generates a sequence of successively better solutions at successively greater computation cost. More specifically, the following variables encode the decisions Xena has to make:

Service placement $P_{Service,\ Address}$ 1 if Service is placed at Address; 0 otherwise

Flow type $F_{FlowId,\ ContentType}$ 1 if FlowId is of type ContentType; 0 otherwise

Service configuration $C_{Service,\ Configuration}$ if Service is configured according to Configuration; 0 otherwise

where the configuration of a service that receives and/or transmits k flows is the k-tuple of the content types of those flows, that is, $Configuration = (ContentType_i)^k$.

Various requirements for semantically correct service delivery are expressed as constraints in the linear program. For example, the constraint

$$\forall i: \sum_j P_{Service_i, Address_j} = 1$$

specifies that a service should be placed on precisely one node. As another example, the following constraint specifies that the flow types which enter or leave a service should match the configuration of the service:

$$\forall i, j, k, l : P_{Service i, Address j} | C_{Service i, Configuration k} \rightarrow F_{Flow(Address j,\ l), FlowType(Service i, Configuration k,\ l)},$$

where the functions $Flow(a, l)$ and $FlowType(s, c, l)$ extract the lth flow and flow type from the node and service configuration, respectively.

Besides the semantics of the service, the optimization problem also has to consider the QoS being delivered. To do this, we need a mathematical representation of QoS that is flexible enough to be used to approximate the more subjective notion of QoS for diverse services. Our QoS definition for value-added services has two parts. First, the user or broker can specify the minimum quality for different aspects of the service; this will result in additional constraints in the linear program. For example, the user could specify a minimum acceptable video quality (frame rate, frame size, resolution) of a Motion JPEG stream; this would be translated by the broker in a constraint on the required bandwidth. Second, the broker or user can specify how QoS can be improved beyond the minimum QoS specified by the additional constraints. This is done by providing an optimization functions of the form

$$\text{maximize} \sum_v w_v \times \text{quality}(v).$$

By appropriately selecting the quality weights, brokers can customize the definition of QoS to the user's needs. For example, assume that a service combines videoconferencing (quality metric is video quality) and access to a set of remote databases (quality metric is response time of queries). By assigning more weight to the bandwidth for videoconferencing than to the communication and computation resources allocated for database access, we can specify that a user prefers high-quality video over fast response time.

An alternative to specifying a quality function is to provide a cost function,

$$minimize \sum_v w_v \times cost(v).$$

This indicates that the user is not interested in improving service quality beyond the minimal quality expressed by the QoS constraints, but instead wants to minimize cost. Again, the weights in the cost function can be chosen to reflect the context of this particular service request.

Given the constraints derived from the semantics of the service, the minimal quality constraints, and the quality/cost function to maximize/minimize, we can call a solver package to perform the actual optimization, described in more detail elsewhere [12]. An example of the use of Xena can be found later.

Resource Brokering Considerations

So far we have focused on the design and implementation of a single broker module. Such a module can be deployed in different ways. For example, a broker could be linked into an application as a library or deployed as a network service that can be invoked by multiple applications. Clearly, in a large-scale environment many brokers will be needed to provide responsive service. Moreover, since brokers incorporate domain knowledge, we envision the deployment of different types of brokers specializing in different application domains (e.g., videoconferencing, distributed games).

Since the optimization problem brokers have to solve is generally NP-hard, an approach based on exact solutions is only appropriate for small to medium-sized problems. Work is in progress on defining heuristics that will make the problem tractable. This approach will necessarily trade quality for performance; our goal is to find high-quality (but not, in general, optimal) solutions at a reasonable cost. To handle large queries, we have to not only speed up individual brokers, but also partition queries across multiple brokers. For example, to handle a query that combines videoconferencing and distributed simulation tasks, it may be best to break it into two smaller queries that are handed off to specialized brokers. Similarly, a multisite query may be broken into WAN and LAN components that are handled by brokers specializing in resource selection for different administrative domains.

Customizable Runtime Resource Management

In this section we discuss how delegates perform customized runtime resource management and describe the delegate runtime environment implemented in Darwin.

Delegates

We use the term *delegate* for a code segment that is sent by applications or service providers to network nodes in order to implement customized management of their data flows. Delegates execute on designated routers, and can monitor network status and adjust resource management on those routers through the *router control interface* (RCI), as shown in Fig. 2. Delegates are a focused application of active networking [13]; they allow users to add customized traffic management and control functions to the network.

A critical design decision for delegates is the definition of the RCI (i.e., the interface delegates use to interact with the environment). Several design goals constrain the definition of the RCI. First, it is highly desirable that the RCI be usable across multiple platforms without being tightly tied to a specific model of resources. In the long run, standardized interfaces for use by control protocols will also be useful for delegates. Second, the RCI should balance the flexibility of

delegate actions with the increase in complexity and cost in routers that support delegates. If the RCI is too restricted, delegates may be rendered virtually useless, while too broad an RCI may make it difficult to efficiently and safely support delegates. Finally, the RCI must support efficient interactions between delegates and the router core under a broad set of conditions. For example, a delegate can detect congestion events by either polling router state or arranging to be notified of events detected by the router core, whichever is more appropriate and efficient for the delegate.

The RCI can be viewed as an instruction set that operates on flows. We have identified the following classes of operations that should be available to delegates:

- *Flow definition*: A flow is a sequence of packets that belong together and should be treated in the same way by the router. Flows are defined by a flow descriptor. Darwin uses the flow descriptor specified by the IETF IntServ specification, although the flow spec can also include an application identifier, which is implemented as an IP option.
- *Resource management*: Delegates can change flows' QoS by adjusting resource allocation and sharing rules. Moreover, they can split or merge flows by adjusting packet classification. Splitting allows different groups of packets in a flow to be treated differently. For example, a delegate can implement selective packet dropping by splitting off a subflow and marking it for discard. A delegate can also elect to receive special control packets by splitting off a subflow and marking it for delivery to itself.
- *Flow redirection*: Delegates can reroute flows and establish tunnels. For example, a delegate might choose to reroute a flow inside a virtual mesh for load balancing reasons, or to direct a flow to a compute server that will perform data manipulations such as compression or encryption.
- *Monitor network status*: This class of operations includes probing of network state (queue occupancy, error flags, etc.). It can also involve posting requests for notification of a small set of specific events, such as crossing of a queue occupancy threshold and occurrence of a failure condition.
- *Communication*: Communication is needed to coordinate activities with peers on other routers or provide network feedback to applications on endpoints.

A fundamental requirement is that potentially concurrent activities by delegates and other network management and control functions maintain a consistent network state. Moreover, we would like these activities to interact efficiently, without complex locking protocols. The hierarchical resource management used in Darwin addresses both requirements. A delegate is associated with a node in the resource tree and is typically only authorized to operate on the flows in "its" subtree, which creates a notion of locality in the organization space. Once the signaling protocol has set up a new flow (node) with an associated delegate, the delegate can operate in relative isolation. Using the resource tree in Fig. 1 as an example, a delegate associated with the SCS node could increase the share of bandwidth available for distributed simulation without having to be concerned about the rest of the system.

The use of delegates also raises significant safety and security concerns. Delegates are in general untrusted, so the router has to ensure that they cannot corrupt state in the router or cause other problems. This can be achieved through a variety of runtime mechanisms (e.g., building a "sandbox" that restricts what the delegate can access) and compile time mechanisms. A related issue is that of security. The two main threats are that delegates affect the use of resources or the behavior of traffic flows that are not under their control. To address this problem, we fall back on the virtual mesh abstraction. Whenever a delegate makes an RCI call, the Darwin kernel makes sure that the call can only affect the use of resources or behavior of traffic flows that are part of the virtual mesh with which the delegate is associated. Access to resources is restricted by giving delegates only access to certain nodes in the resource trees of the output ports of the router (see next section). Access to traffic is limited by making sure that the filter specification of an RCI call covers only a subset of the traffic which flows within the delegate's virtual mesh. These security checks are implemented through the use of access control lists, similar to the access control lists used to protect the contents of file systems. These access control lists are created by the local resource manager when a delegate is created.

Implementation

Our current framework for delegates is based on a "standard" Java Virtual Machine (JVM). We use the JVM from Sun Microsystems, which is available on many platforms, including NetBSD and FreeBSD. This environment gives us acceptable performance, portability, and safety features inherited from the language. Delegates are executed as Java threads inside the virtual machine sandbox. Delegates can run with different static priorities, although a more controlled environment with real-time execution guarantees is desirable.

The RCI is implemented as a set of native methods that call the local resource manager, which runs in the kernel. Communication among delegates and between delegates and endpoints uses sockets, built on top of the standard java.net classes. A more detailed description of delegates, including a set of examples, can be found elsewhere [14].

Hierarchical Scheduling

The virtual mesh offers an elegant abstraction since it allows the user to use and manage a set of resources as if they were a dedicated infrastructure, even though the resources are part of a larger shared infrastructure. However, the abstraction is meaningless unless the network can guarantee that the service provider or application will be able to use and control the resources assigned to it. In Darwin, the resource isolation and sharing needed to realize virtual meshes is implemented by the H-FSC scheduler.

Scheduling Algorithm

As discussed earlier, from a resource management point of view, a service-oriented network can be viewed as a hierarchy of virtual meshes, and each virtual mesh represents the set of resources that are managed by an entity such as a service provider or an application. For each individual physical resource, sharing and thus contention exist at multiple levels: at the physical resource level among multiple service providers, at the service provider level among lower-level service providers or organizations, and at the application level among individual flows. These relationships can be represented by a resource tree: each node represents one entity (resource owner, service provider, application), the slice of the virtual resource allocated to it, the traffic aggregate supported by it, and the policy of managing the virtual resource; each arc represents the virtual resource owner/user relationships. An example of a resource tree is shown in Fig. 1.

The ability to customize resource management policies at all sharing levels for a resource is one of the key requirements and distinctive features of service-oriented networks. The challenge is to design scheduling algorithms that can simultaneously satisfy diverse policies set by different entities in the resource management tree. In Darwin we use

the H-FSC scheduling algorithm [6]. An important feature of H-FSC is that entities sharing the resource at different levels in the resource hierarchy can manage their share of the resources *independently*. For example, in the resource hierarchy shown in Fig. 1 the two service providers use radically different resource management policies: service provider 2 supports the IETF Intserv QoS model (guaranteed and controlled load) for individual traffic streams, while the more sophisticated service provider 1 supports organization-based QoS, where the organizations and applications can specify the dynamic sharing relationship for their traffic streams. Moreover, once the resource sharing relationship between providers 1 and 2 is set up (weighted sharing with 100 Mb/s and 55 Mb/s for each), the two providers can independently change resource use without affecting the QoS properties of traffic supported by the other provider. For example, provider 1 can admit new flows or change the resource allocation for existing flows without having to coordinate with provider 2, as long as its stability conditions are met.

Implementation

Besides the scheduler, the implementation of the H-FSC scheduler in Darwin also needs a packet classifier and an API for signaling protocols. The classifier classifies incoming data packets to a specific flow and packet queue. This is done by matching a packet's header against a set of known flow descriptors. To support traffic flows of various granularities, we have devised a highly flexible classification scheme. Each flow is described by a nine-parameter flow descriptor: source IP address, source address prefix mask, destination IP address, destination address prefix mask, protocol number, source port number, destination port number, application-specific ID (carried as an option in the IP header), and application-specific ID prefix mask. For each parameter, zero denotes "don't care." Using this scheme, flows of various granularity, such as end-to-end TCP connections, aggregates of traffic between networks (using network prefixes in the flow descriptors), and HTTP, FTP, and TELNET services can be specified. With the application-specific ID, we can even subdivide an end-to-end traffic flow into application-specific subflows, such as the different frame types in an MPEG flow.

A control application programming interface (API) is exported by the scheduler and classifier, and used to create and manage data structures in the scheduler and classifier. It is available to delegates and Beagle through the RCI, as discussed earlier.

The processing overhead associated with classification and queuing in our implementation is fairly low. Classification overhead is 3 µs/packet with caching on a 200 MHz Pentium Pro system. Average queuing overhead is around 9 µs when there are 1000 flows in the system. This low overhead allows us to easily support an actual data speed of 100 Mb/s in our prototype network testbed. A more detailed analysis of the performance of the H-FSC implementation can be found in [6] and some experimental results in [15].

The Beagle Resource Allocation Protocol

The Beagle resource allocation protocol provides support for the customizable resource allocation model of Darwin. While traditional signaling protocols such as RSVP [8] and ATM private network-to-network interface (PNNI) operate on individual flows, Beagle operates on virtual meshes. We will use the application example in Fig. 3 to illustrate the various tasks Beagle performs. The example is discussed in more depth later.

Beagle Design

On the input side, Beagle interfaces with Xena to obtain the virtual mesh specification generated by Xena. The virtual mesh is described as a list of flows and delegates, plus resource sharing specifications that describe how flows within a mesh share resources among them. The example virtual mesh in Fig. 3b shows two video flows and two distributed interactive simulation flows. Each flow is specified by a *flow descriptor*, described earlier, and information such as a *tspec* and a *flowspec* object (used to characterize the properties of a traffic flow and the corresponding reservation), as in the IETF IntServ working group model. A delegate is characterized by its resource requirements (CPU, memory, and storage), its runtime environment, and a list of flows the delegate needs to manipulate. The delegate runtime environment is characterized by a code type (e.g., Java, Active-X) and *runtime type* (e.g., JDK 1.0.2, WinSock 2.1). The virtual mesh typically also includes a number of designated routers that identify the mesh core. In the example, Aspen and Timberline are the designated routers.

After receiving a request, Beagle issues a sequence of flow setup messages to the different nodes in the mesh, each specifying the total resources needed on a link, plus a resource sharing specification (described below). Since Beagle has access to the resource specification for the full mesh, it has many options for setting up the mesh. In [16] we compare a traditional per-flow setup with a core-based setup, in which the different segments of the core are set up in parallel, followed by individual flow setups initiated by the senders or receivers that rendezvous with the core in designated routers. For meshes in which flows share resources in the core, core-based setup is more efficient because it can allocate resources for flow aggregates in the core.

On each node, Beagle passes a resource tree (Fig. 3c) to the *local resource manager* to allocate resources for flows using an interface similar to the one described earlier. Beagle also establishes delegates onto switch nodes (for resource management delegates), or computation and storage nodes (for data processing delegates). For each delegate request, Beagle locates the appropriate runtime environment, initializes the local resource manager handles and flow reservation state, and instantiates the delegate. The handles allow the delegate to give resource management instructions to the local resource manager for the flows associated with it.

Beagle requests have to identify the entity that invoked Beagle. This allows the local resource manager to verify that the Beagle request is valid (i.e., the caller is allowed to allocate the resources, control the traffic, and create the delegates specified in the request). The local resource manager will also set up the access control lists for any delegates it creates so the delegate's actions can be verified during execution. Authentication and authorization for Beagle requests are currently being implemented.

Resource Sharing

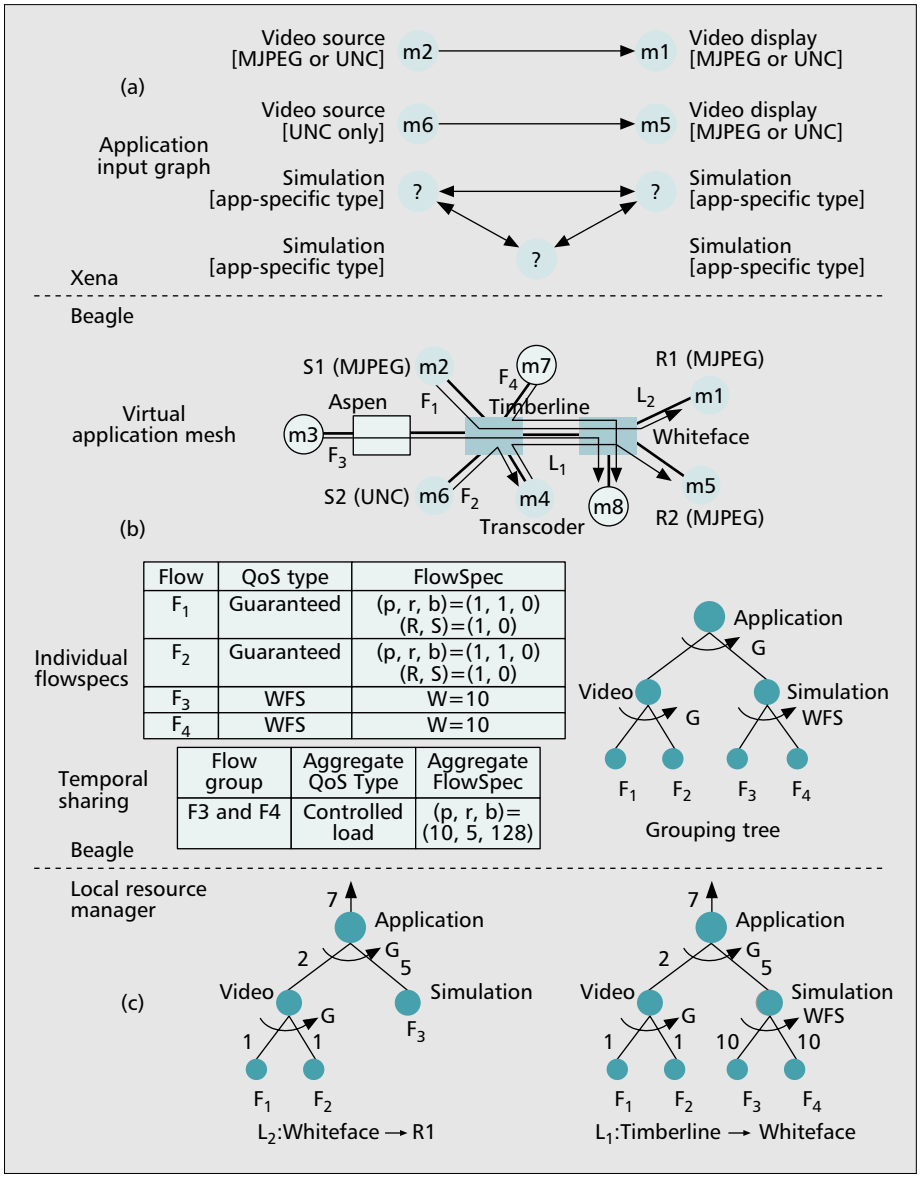
Beagle supports two forms of resource sharing: hierarchical and temporal.

In the previous section we described how dynamic resource sharing can be controlled and customized by specifying an appropriate resource tree for each resource. This could be achieved by having applications or Xena specify the resource trees to Beagle so that it can install them on each node. There are two problems with this approach. First, how one specifies a resource tree is network-specific, and it is unrealistic to expect applications and brokers to deal with this heterogeneity. Second, applications and brokers do not specify each phys-

ical link; instead, they use virtual links that may represent entire subnets. Beagle uses the *hierarchical grouping tree* abstraction to deal with both problems: it is an abstract representation of the sharing hierarchy that can be mapped onto each link by Beagle. Once Beagle knows the actual flows that share a particular physical link in the network, Beagle prunes the hierarchical grouping tree, eliminating flows which do not exist at that link. To deal with network heterogeneity, interior nodes in the hierarchical grouping tree have generic QoS service types associated with them instead of network-specific sharing specifications. The leaf nodes of the grouping tree represent flows whose QoS requirements are expressed by individual flowspecs. Service-specific rules describe how child node flowspecs are aggregated into parent node flowspecs in deriving a physical link resource tree from the grouping tree. This involves converting flowspecs at each node into appropriate low-level scheduler-specific parameters, such as a weight for hierarchical weighted fair share schedulers or a service curve for the H-FSC scheduler [6]. More details can be found in [16]; Fig. 3b gives an example.

There are often resource sharing opportunities on time scales larger than can be expressed in tspecs and flowspecs. For example, a conferencing application may ensure that at most two video streams are active at any time, or an application may want to associate an aggregate bandwidth requirement for a group of best-effort flows. We will call this *temporal* or *sequential* sharing (i.e., a set of flows share the same resource over time). Applications and resource brokers can specify this application-specific information by handing Beagle temporal sharing objects that list sets of flow combinations and their associated aggregate flowspecs. Beagle can then use this information to reduce resource requirements for a group of flows sharing a link. The temporal sharing object is similar in spirit to the resource sharing approach used in the Tenet-2 scheme [17], and generalizes RSVP's notion of resource reservation styles. RSVP limits aggregation to flows within a multicast session and restricts aggregation (e.g., least upper bound) of flow specs, while in Beagle the arbitrary flows within an application mesh can be grouped using any aggregate flowspec. As an example, in Fig. 3b the distributed interactive simulation application associates an aggregate controlled load service flowspec with the two simulation flows.

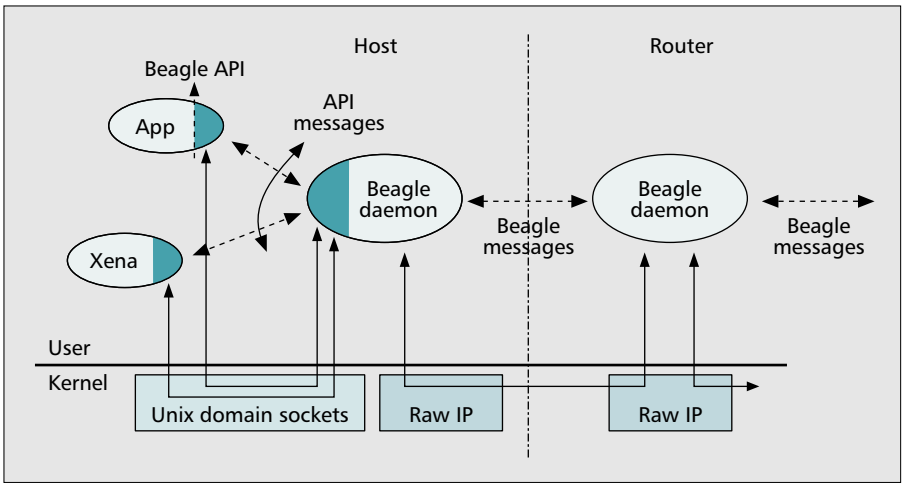
The hierarchical grouping tree and temporal sharing objects offer complementary and orthogonal ways of tailoring resource allocation within the mesh. The example in Fig. 3b shows the use of both sharing objects. The resulting link resource subtrees at links L_1 and L_2, assuming the use of hierarchical weighted fair queuing schedulers [5], are shown in Fig. 3c.



■ Figure 3. *Handling an application service request in Darwin.*

Implementation

A prototype of the Beagle resource allocation protocol has been implemented for FreeBSD and NetBSD PC-based routers (Fig. 4). The implementation is based on the RSVP implementation distributed by ISI (available from ftp://ftp.isi.edu/rsvp/release). Applications or Xena can invoke Beagle through calls such as "create mesh" or "extend mesh." The API (shown by the shaded portions in Fig. 4) is a library compiled as part of the application. It communicates with the Beagle daemon through UNIX domain sockets. The API also has a callback mechanism used to asynchronously notify applications about various events such as setup success/failure and incoming requests. The Beagle daemon communicates with other Beagle daemons using raw IP. The current Beagle prototype includes support for both temporal and hierarchical sharing. On 200 MHz Pentium routers running FreeBSD 2.2.5, the Beagle throughput is about 425 flow setups/s. This is comparable to connection setup times reported for various asynchronous transfer mode



■ Figure 4. *The Beagle prototype and Beagle API implementation.*

(ATM) switches. However, we expect improvement in these results by optimizing the implementation.

Darwin System Demonstration

In this section we present an example demonstrating how the Darwin resource management mechanisms work together in support of a remote collaboration application executing on a testbed.

Darwin Testbed

The topology of the testbed is shown in Fig. 3b. The three routers represented by rectangles are Pentium II 266 MHz PCs running NetBSD 1.2D. End systems m1–m8 are Digital Alpha 21064A 300 MHz workstations running Digital UNIX 4.0. All links are full-duplex point-to-point Ethernet links configurable as either 100 Mb/s or 10 Mb/s.

The Application Scenario

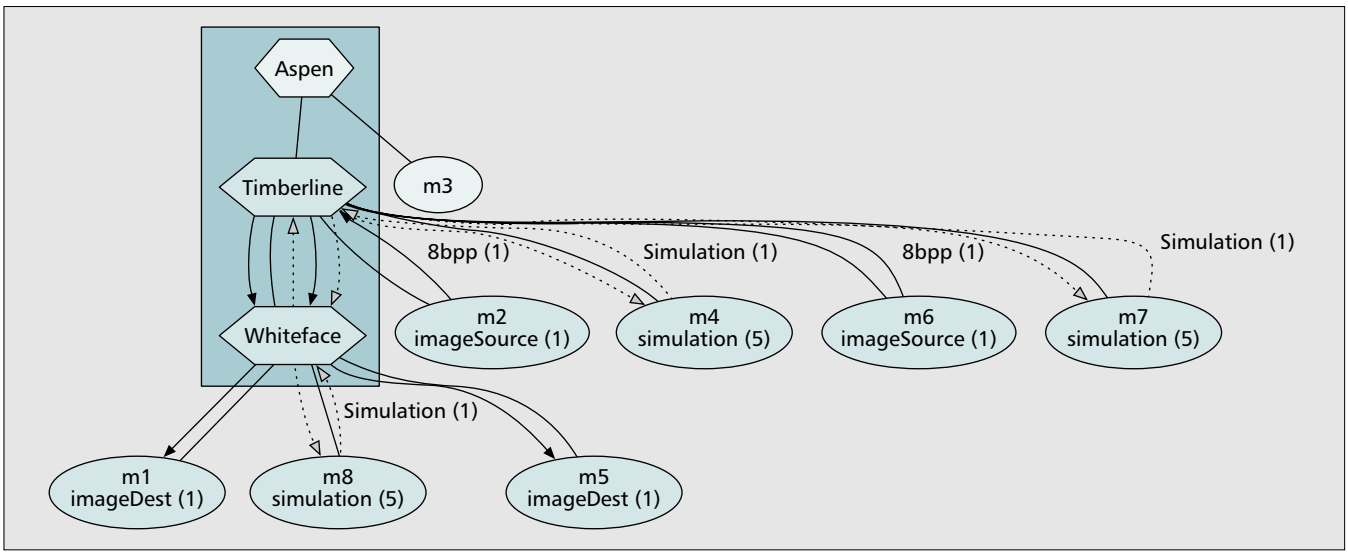
Consider an application in which four scientists communicate via a videoconferencing tool that uses motion JPEG coders, and collaborate on a distributed simulation that runs over our testbed. This scenario is shown in Fig. 3. In this case, the videoconferencing consists of two unicast video flows between desktop systems, and the distributed simulation consists of a three-node distributed fast Fourier transform (FFT), which generates very bursty traffic.

Using our testbed (Fig. 3b), the machines involved in the videoconference are m1, m2, m5, and m6. Source m2 can send either 8-bit uncompressed video or Motion JPEG (MJPEG) compressed video; source m6 can send only uncompressed video. The users do not care which nodes are used for the distributed simulation, so they leave it up to "the network" (more specifically, Xena) to select the most appropriate simulation servers. In this section we use a combination of performance results and screen shots of the running system to demonstrate the operation of Darwin.

Setup Using Xena and Beagle

Figure 3a shows the abstract resource mesh supplied to Xena. Since the scientists are physically located at their machines, the application provides to Xena specific network addresses for the video endpoints, (m1, m2, m5, m6). The nodes are also annotated by the requested service types (video source, video display). This flow specification describes only the connectivity between the nodes; the flow's exact QoS parameters are left unspecified. For the distributed simulation, the application does not specify which nodes should participate, so these addresses will have to be supplied by Xena's placement algorithm.

We consider two scenarios under which Xena has to generate a virtual mesh layout for the given application input. In the first *abundant bandwidth* scenario, bandwidth is plentiful on all the links, but computational cycles are relatively scarce. In the second *scarce bandwidth* scenario, the links are heavily loaded, but computational resources are relatively abundant. Xena accounts for these loads by adjusting the costs of various resources. Costs are assigned according to the quantity of the resource desired and its availability. For example, the cost for raw video flows is higher than for MJPEG flows because



■ Figure 5. *The Xena virtual mesh layout in the abundant bandwidth scenario.*

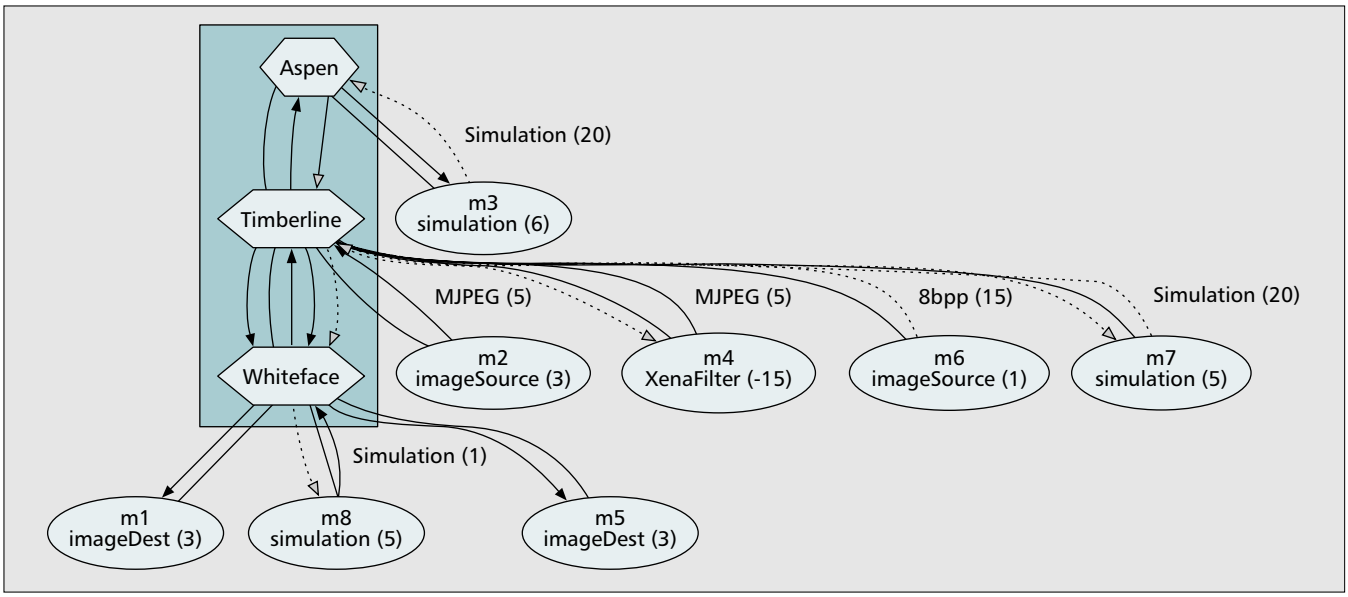


Figure 6. *The Xena virtual mesh layout in the scarce bandwidth scenario.*

raw video requires more bandwidth; also, the cost for a given quantity of some resource may be higher than otherwise if the level of contention is high for that resource. Xena achieves this latter effect by assigning an overlap penalty when more than one service is assigned to the same endpoint.

Figures 5 and 6 show the virtual mesh layouts generated by the running Xena prototype. In the abundant bandwidth scenario, Xena selects uncompressed video encoding, which requires more bandwidth but is computationally less expensive. It places the simulation tasks on nodes m4, m7, and m8, avoiding the video endpoints (m1, m2, m5, and m6). It also places two simulation nodes (m4 and m7) on endpoints attached to the same router, thus minimizing the use of link bandwidth. In contrast, in the scarce bandwidth scenario, Xena selects the less bandwidth-intensive MJPEG video encoding to reduce the use of expensive communication resources. To accommodate m2, which supports only uncompressed video, it inserts a video transcoder into the mesh at m4. The transcoder is placed close to the video source to minimize the use of expensive link bandwidth. The transcoder is a hardware MJPEG compressor that provides the ability to dynamically control compression quality; note that higher-quality video streams require more bandwidth. In order to opportunistically use available bandwidth, Xena also instantiates a control delegate on timberline which is responsible for monitoring the available bandwidth and setting the compressor's quality knob appropriately. Xena distributes the simulation tasks on the remaining nodes (m3, m7, and m8), thus avoiding the overlap penalty for computations.

Once it has a solution, Xena generates a mesh specification similar to the one shown in Fig. 3b. Beagle allocates the resources as described above, and also sets up the transcoder and the control delegate on timberline. Figure 7 is a screen shot generated using the H-FSC scheduler user interface program that shows the resource sharing tree setup by Beagle for the timberline → whiteface link. The application is the right subtree (root number 42), and it uses separate subtrees for the video streams and FFT data flows.

Execution

We present experimental results by executing the video and FFT applications under three scenarios, one in the abundant bandwidth scenario and two in the scarce bandwidth scenario. In all experiments, 70 percent of the link bandwidth is allocated to the application, and the remainder is used by a UDP cross-traffic flow.

In the abundant bandwidth scenario, all links are configured to run at 100 Mb/s, and the UDP flow uses 30 Mb/s.

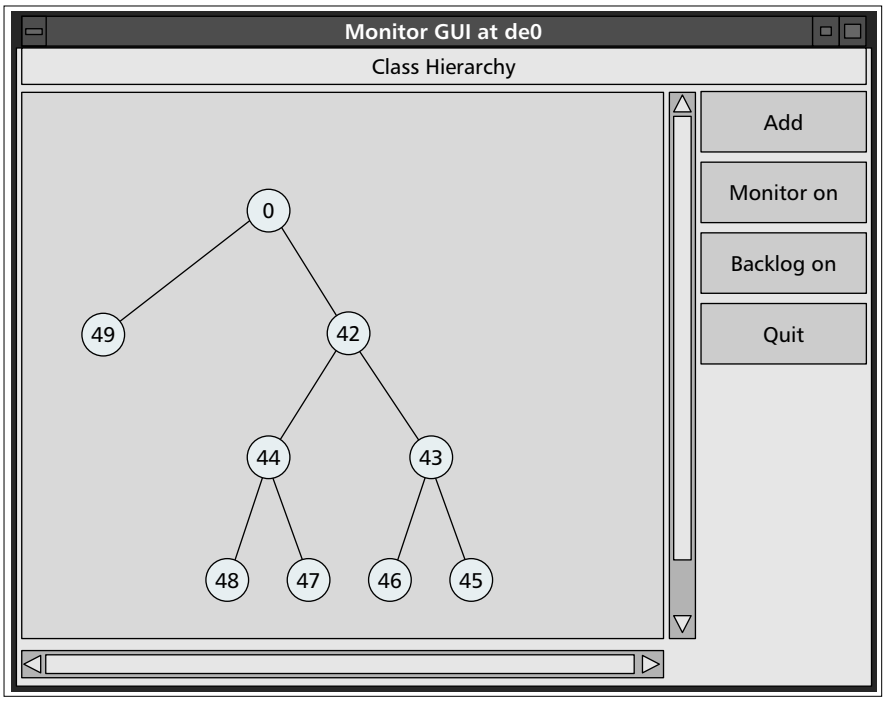


Figure 7. *The resource tree for the timberline → whiteface link.*

Both uncompressed video flows are allocated a bandwidth of 18 Mb/s each (320 x 240 8-bit pixels at 30 frames/s), and the remaining 34 Mb/s is allocated to FFT flows. In this scenario, there is enough bandwidth so both video streams can be received without experiencing packet loss. This scenario has the highest video quality.

In the scarce bandwidth scenario, the link between timberline and whiteface is slowed down to 10 Mb/s, and the UDP flow is allocated 3 Mb/s. Each of the two MJPEG compressed video flows crossing this link are guaranteed 1 Mb/s (i.e., the bandwidth for the lowest setting of the JPEG compression quality parameter), and the rest of the bandwidth is allocated to FFT flows. Without a control delegate, both video flows are guaranteed only a minimum bandwidth of 1 Mb/s, and therefore the received video quality is the lowest. Using a control delegate allows the video flows to opportunistically take advantage of available bandwidth by adjusting quality on a dynamic basis. Figure 8 shows a histogram of the received frame quality. The x-axis represents the video quality parameter that is the input to the JPEG compression card. As seen from the figure, the majority of the frames are received with either maximum quality of 100 (received when the FFT is in its computation phase) or minimum quality of 0 (when the FFT is in its communication phase). Frames received with the other quality settings reflect the ramp up and ramp down behavior of the control delegate as it tracks the available bandwidth.

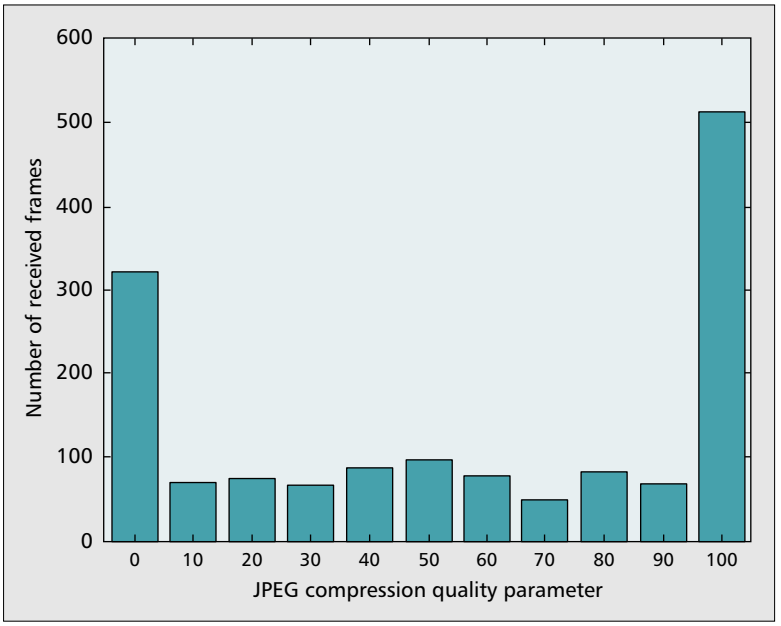


Figure 8. *A histogram showing the distribution of received JPEG quality.*

Figure 9 shows the bandwidth sharing within the example application on the timberline → whiteface link under the three experimental scenarios. In all plots, the dark grey portion depicts the aggregate bandwidth used by the two FFT flows and the light grey portion shows the aggregate bandwidth used by the two video flows. We see that the FFTs exhibit very bursty communication patterns. Figure 9a shows the sharing in the abundant bandwidth case: there is ample bandwidth available for both the video and FFT data streams. Figure 9b shows the corresponding plot under the scarce bandwidth scenario without using a control delegate. If the video source tried to transmit at a high quality level during FFT idle periods, it would not be able to adapt effectively because it cannot learn about the excess bandwidth quickly enough. As a result, it always sends at the lowest quality level, even though the FFT is idle most of the time. Figure 9c shows the sharing when a control delegate is used to control the operation of the transcoder. The control delegate can quickly detect the idle periods in the FFT communication patterns and increase the MJPEG compression quality parameter accordingly. This enables the transcoded video flow to opportunistically take advantage of available bandwidth.

Related Work

There has recently been a lot of work as part of the Xbind [3] and TINA [18] efforts to define a service-oriented architecture for telecommunications networks. There are several differences between them and Darwin. First, services envisioned by Xbind and TINA are mostly telecommunications-oriented. The value-added services described in this article integrate computation, storage, and communication resources. Second, the value-added services in their context are usually restricted to the control plane (e.g., signaling). Darwin supports customized services in the data plane (controlled sharing of resources, processing, and storage) and the control plane (signaling and resource brokering). Finally, while the focus of both TINA and Xbind is on developing an open object-oriented programming model for rapid creation and deployment of services, the focus of Darwin is on developing specific resource management *mechanisms* that can be *customized* to meet service-specific needs. While

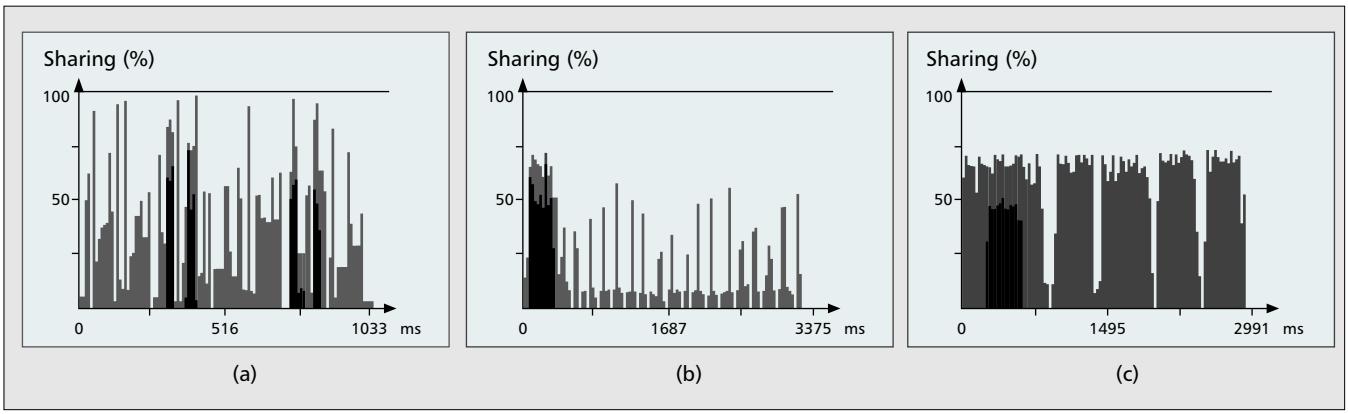


Figure 9. *Bandwidth sharing within the example application: a) abundant bandwidth scenario; b) scarce bandwidth scenario without the control delegate; c) scarce bandwidth scenario with the control delegate.*

Xbind and TINA have so far primarily been used as a development framework for traditional ATM and telecommunications network management mechanisms, they could potentially also be used as a basis for the development of customizable resource management mechanisms.

Another very active area of research is the development and deployment of "computational grids" [19, 20], distributed pools of resources available for use by distributed applications. The vision underlying computational grids is similar to that of Darwin, namely that the most effective way of satisfying diverse user needs is to provide an open programmable infrastructure that users can customize for their use by installing the appropriate software. The main difference between Darwin and computational grids is the class of applications they have focused on (i.e., network-intensive vs. compute-intensive applications). This has resulted in a very different research agenda: managing and customizing the use of network resources vs. compute and storage resources.

Over the past decade much work has gone into defining QoS models and designing associated resource management mechanisms for both ATM and IP networks [2]. This has resulted in specific QoS service models for both ATM and IP [21]. This has also resulted in the development of QoS routing protocols and signaling protocols [8]. A closely related issue being investigated in the IP community is link sharing [4], the problem of how organizations can share network resources in a preset way while allowing the flexibility of distributing unused bandwidth to other users. Darwin differs from these efforts in several aspects. First, while most of this work focuses on communication services, Darwin addresses both bitway and value-added service providers. Second, most QoS models only support QoS on a per-flow basis. Exceptions are the concept of virtual path and virtual circuit in ATM, and the IP DiffServ model [9], but these efforts are very restricted in either the type of hierarchy they support or the number of traffic aggregates for which QoS can be provided. In contrast, Darwin uses virtual meshes to define service-specific QoS and supports controlled resource sharing among dynamically defined traffic aggregates of different granularities. Finally, while these efforts provide resource management mechanisms on the space, time, and organizational dimensions, the mechanisms operate largely in an isolated and uncoordinated fashion. On the other hand, Darwin takes an integrated view OF resource management along these three dimensions.

While Darwin is primarily a networking project, guaranteeing end-to-end QoS properties also requires QoS support on the endpoints. There has been a lot of research in real-time operating systems [22]; in Darwin we assume that heavily loaded endpoints (typically the servers) will have real-time operating system support so they can give appropriate service to users. A number of projects have also looked at the problem of providing end-to-end QoS guarantees [23, 24]. These research efforts typically develop mechanisms that support the precise QoS needs of specific application classes, while Darwin has focused on creating an infrastructure in which diverse services and applications with different QoS requirements can coexist.

Finally, *active networks* have recently attracted a lot of attention. In an active network, packets carry code that can change the behavior of the network [13]. The Darwin project touches on this concept in two ways. First, service delegates are an example of active packets, although a very restricted one: delegates are typically downloaded to a specific node at service invocation time, and remain in action for the duration of the service. Second, Darwin's facilities for managing both computation and communication resources via virtual meshes can help solve key resource allocation problems faced by active networks.

Summary

We have designed a resource management system called Darwin for service-oriented networks that takes an integrated view of resource management along space, time, and service dimensions. The Darwin system consists of four interrelated resource management mechanisms: resource brokers called Xena, runtime resource management using delegates, H-FSC hierarchical packet scheduling, and a resource allocation protocol called Beagle. The key property of all these mechanisms is that they can be customized according to service-specific needs. The first three mechanisms (Xena, delegates, and H-FSC) operate on different time scales, which also influences the complexity and scope of resource management decisions they make, and thus their role in the system. While the Darwin mechanisms are most effective when they work together in Darwin, each mechanism can also be used in a plug-and-play fashion in more traditional QoS architectures (e.g., Beagle for RSVP, Xena for resource brokers, and hierarchical scheduling for traffic control). We have a proof-of-concept implementation of the Darwin system and preliminary experimental results to validate the architecture.

The Darwin prototype described in this article implements the vision and demonstrates some of the possibilities, but much work remains to be done. Future versions will be far more scalable, in terms of both routing in large topologies and aggregate processing of large numbers of flows. Security features are currently rudimentary, and explicit authentication, authorization, and encryption methods remain to be incorporated. Hierarchical resource management has been implemented only for network components, and must be extended to computation and storage resources. Finally, a number of topics, while important to a complete network, are beyond the scope of the current project: tools for service creation, mechanisms for automated discovery of resources, and detailed accounting of resource commitment and use.

References

[1] R. Braden, D. Clark and S. Shenker, "Integrated Services in the Internet Architecture: An Overview," Internet RFC 1633, June 1994.

[2] D. Ferrari and D. Verma, "A Scheme for Real-Time Channel Establishment in Wide-Area Networks," *IEEE JSAC*, vol. 8, no. 3, Apr. 1990, pp. 368–79.

[3] A. Lazar, K.-S. Lim and F. Marconcini, "Realizing a Foundation for Programmability of ATM Networks with the Binding Architecture," *IEEE JSAC*, vol. 14, no. 7, Sept. 1996, pp. 1214–27.

[4] S. Floyd and V. Jacobson, "Link-Sharing and Resource Management Models for Packet Networks," *IEEE/ACM Trans. Net.*, Aug. 1995, vol. 3, no. 4, pp. 365–86.

[5] J. C. R. Bennett and H. Zhang, "Hierarchical Packet Fair Queuing Algorithms," *Proc. SIGCOMM '96 Symp. Commun. Architectures and Protocols*, Aug. 1996, Stanford, CA, pp. 143–56.

[6] I. Stoica, H. Zhang, and T. S. E. Ng, "A Hierarchical Fair Service Curve Algorithm for Link-Sharing," Real-Time and Priority Service, *Proc. SIGCOMM '97 Symp. Commun. Architectures and Protocols*, Cannes, France, Sept. 1997, pp. 249–62.

[7] L. Delgrossi and D. Ferrari, "A Virtual Network Service for Integrated-Services Internetworks," *Proc. 7th Int'l. Wksp. Network and OS Support for Digital Audio and Video*, St. Louis, MO, May 1997, pp. 307–11.

[8] L. Zhang *et al.*, "RSVP: A New Resource Reservation Protocol," *IEEE Commun. Mag.*, vol. 31, no. 9, 1993, Sept. 1993 pp. 8–18.

[9] S. Blake *et al.*, "An Architecture for Differentiated Service," IETF RFC 2475, Dec. 1998.

[10] K. Lim, "A Network Architecture for Virtual Private Networks with Quality of Service," Master's thesis Info. Networking Inst., Carnegie Mellon Univ., Mar. 2000.

[11] M. Berkelaar, "lp_solve: A Mixed Integer Linear Program Solver," ftp://ftp.es.ele.tue.nl/pub/lp_solve/, Sept. 1997.

[12] P. Chandra *et al.*, "Network Support for Application-Oriented Quality of Service," *Proc. 6th IEEE/IFIP Int'l. Wksp. QoS*, Napa, CA, May 1998, pp. 187–95.

[13] D. Tennenhouse and D. Wetherall, "Towards and Active Network Architecture," *Comp. Commun. Rev.*, vol. 26, no. 2, Apr. 1996, pp. 5–18.

[14] J. Gao *et al.*, "A Programmable Router Architecture Supporting Control Plane Extensibility," *IEEE Commun. Mag.*, vol. 38, no. 3, Mar. 2000, pp. 152–59.

[15] P. Chandra *et al.*, "Darwin: Customizable Resource Management for Value-Added Network Services," *6th Int'l. Conf. Network Protocols*, Austin, TX, Oct. 1998, pp. 177–88.

[16] P. Chandra, A. Fisher, and P. Steenkiste, "A Signaling Protocol for Structured Resource Allocation," *IEEE INFOCOM '99*, New York, NY, Mar. 1999, pp. 522–33.
[17] A. Gupta, "Resource Sharing in Multi-party Realtime Communications," *Proc. INFOCOM '95*, Boston, MA Apr. 1995, pp. 1230–37.
[18] F. Dupuy, C. Nilsson, and Y. Inoue, "The TINA Consortium: Toward Networking Telecommunications Information Services," *IEEE Commun. Mag.*, vol. 33, no. 11, Nov. 1995, pp. 78–83.
[19] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *Int'l. J. Supercomp. Apps.*, 1997, vol. 11, no. 2, pp. 115–28.
[20] A. Grimshaw, W. Wulf and The Legion Team, "The Legion Vision of a Worldwide Virtual Computer," *Commun. ACM*, Jan. 1997, vol. 40, no. 1, pp. 39–45.
[21] S. Shenker, C. Partridge, and R. Guerin, "Specification of Guaranteed Quality of Service," IETF RFC 2212, Sept. 1997.
[22] K. Nahrstedt and J. M. Smith, "A Service Kernel for Multimedia Endstations," *Multimedia: Adv. Teleservices and High-Speed Commun. Architectures*, vol. 2, no. 1, 1994, pp. 53–67.
[23] C. Lee *et al.*, *Proc. IEEE Real-Time Systems Symp.*, Dec. 1999, pp. 315–26.
[24] K. Nahrstedt, H.-H. Chu and S. Narayan "QoS-Aware Resource Management for Distributed Multimedia Applications," *J. High-Speed Networks*, Special Issue on Multimedia Networking, vol. 8, no. 3–4, 1998, pp. 227–55.

Biographies

PRASHANT R. CHANDRA received his B.E in electronics engineering from Bangalore University in 1991, M.S in computer engineering from West Virginia University in 1994, and Ph.D. in computer engineering from Carnegie Mellon University in 2000. He is currently a network architect at Intel Corporation. His research interests are in the areas of programmable networks, signaling protocols, and traffic engineering.

YANG-HUA CHU is a Ph.D. student in the Computer Science Department at Carnegie Mellon University. He received B.S. and M.Eng. degrees from the Massachusetts Institute of Technology in 1996 and 1997, respectively. His research interests are in multicast and content distribution.

ALLAN FISHER received a Ph.D. in computer science at Carnegie Mellon in 1984. He was a Churchill Scholar at the University of Cambridge from 1978 to 1979. He received the A.B. in chemistry from Princeton University in 1978. He has been on the computer science faculty at Carnegie Mellon since 1984 and was Associate Dean for Undergraduate Studies 1988–1998. He has researched and published widely in the area of high-performance computing and networking.

JUN GAO received his B.S. degrees in engineering physics and computer science in 1995 from Tsinghua University, Beijing, China, an M.S. degree in nuclear engineering in 1997 from University of Virginia, and an M.S. degree in computer science in 1999 from Carnegie Mellon University. He is currently a Ph.D. candidate in the Computer Science Department at Carnegie Mellon. His research interests include network resource management mechanisms and customizable Internet services.

COREY KOSAK received a B.A. in computer science from Harvard University in 1991 and an M.S. in computer science from Carnegie Mellon University in 1993. He is currently a Ph.D. candidate in computer science at Carnegie Mellon. His research focuses on developing algorithms for high-quality resource allocation in advanced services networks.

T. S. EUGENE NG received his B.S. in computer engineering from the University of Washington in 1995 and his M.S. in computer science from Carnegie Mellon University in 1998. He is currently a Ph.D. candidate in computer science at CMU. His thesis research focuses on developing a third-party network service to enable connectivity across Internet networks of heterogeneous address spaces, and performance optimization techniques in a wide range of third-party network services.

PETER STEENKISTE (prs@cs.cmu.edu) is an associate professor in the School of Computer Science and the Department of Electrical and Computer Engineering at Carnegie Mellon University. He received the degree of Electrical Engineer from the University of Gent in Belgium in 1982, and M.S. and Ph.D. degrees in electrical engineering from Stanford University in 1983 and 1987. His research interests are in the area of network support for electronic services.

EDUARDO S. C. TAKAHASHI received an M.S. degree in electrical and computer engineering from Carnegie Mellon University in 1999, an M.S. degree in electrical engineering from Universidade de Sao Paulo, Brazil, in 1995, and a B.S. degree in electrical engineering from Instituto Tecnologico de Aeronautica, Brazil, in 1989. His areas of interest include media streaming, resource management, QoS provisioning, and wireless, mobile, active, and pervasive networking.

HUI ZHANG [M'95/ACM'95] is the Finmeccanica Associate Professor at the School of Computer Science of Carnegie Mellon University. He received a B.S. in computer science from Beijing University in 1988, an M.S. in computer engineering from Rensselaer Polytechnic Institute in 1989, and a Ph.D. in computer science from the University of California at Berkeley in 1993. His research interests are in scalable solutions for QoS and value-added services over the Internet. He received the National Science Foundation CAREER Award in 1996 and the Alfred Sloan Fellowship in 2000.