Darwin: Customizable Resource Management for Value-Added Network Services

Prashant Chandra†, Allan Fisher‡, Corey Kosak‡,
T. S. Eugene Ng‡, Peter Steenkiste‡, Eduardo Takahashi†, Hui Zhang‡
† Electrical and Computer Engineering, ‡ School of Computer Science
Carnegie Mellon University
Email: {prashant,alf,kosak,eugeneng,prs,takahasi,hzhang}@cs.cmu.edu

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, 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 organizations sharing physical resources, and be customizable so services and applications can tailor resource usage to optimize their performance.

The Darwin project is developing a set of customizable resource management mechanisms that support value-added services. In this paper we present these mechanisms, describe their implementation in a prototype system, and describe the results of a series of proof-of-concept experiments.

1. Introduction

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 IntServ to service frameworks like Xbind, they share the overall goal of evolving the Internet service model from what is essentially a bitway pipe to a sophisticated infrastructure capable of supporting novel advanced services.

In this paper, 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, service providers will be able to support sophisticated value-added services such as intelligent caching, video/audio transcoding and mixing, 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; the most basic service is provided by bitway and computation providers. 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 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 paper, we focus on the resource management architecture and algorithms for such a network. In particular, we observe that 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 different types of resources can be exploited by value-added service providers to achieve higher efficiency. For example, by using compression techniques, one can make tradeoffs between network bandwidth and CPU cycles. Similarly, storage can sometimes be traded for bandwidth. 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 tradeoffs based on their own notion of quality. Finally, it must not only be possible to make the above tradeoffs at startup, but also to adapt resource management to the changing network conditions 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 customization is needed: space (what network resources are needed), time (how the resources are applied over time), and organization (how the resources are shared with other organi-

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

zations). Additionally, the mechanisms along all three dimensions need to to be integrated, so that they can leverage off one another.

The Darwin project is developing a comprehensive set of customizable resource management tools that support value-added services. In this paper we first identify key resource management requirements (Section 2). We then outline and motivate the Darwin architecture (Section 3). In Sections 4 through 7 we describe each of the four components of the Darwin architecture in more detail, and we illustrate their operation using several proof of concept experiments. We demonstrate the integrated operation of Darwin in Section 8, discuss related work in Section 9, and summarize in Section 10.

2. Customizable Resource Management

2.1. Resource Management Requirements

The attributes of value-added services described in the Introduction suggest several requirements for resource management. We organize these requirements into the following classes: resource management in space, across organizational boundaries, and in time.

- *space*: Services need to be able to allocate a rich set of resources (links, switch capacity, storage capacity, compute resources, etc.) to meet their needs. Resources should be allocated in a coordinated fashion, since there are often dependencies between resources, and there should be support for global optimization of resource use.
- *organizations*: 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 organizations.
- *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 because 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 what resources they allocate, how they share resources with other organizations, and how they adapt their resource use over time.

2.2. Darwin Resource Management Mechanisms

While it may seem attractive to have a single resource management mechanism that meets the above requirements, there are several reasons why an integrated set of mechanisms is preferable. First, resource management activities cover a wide dynamic range in both time and space: some tasks are invoked rarely (e.g. only at application setup time), whereas others are invoked very often (e.g. on every packet passing through a router); different activities also have different costs. Similarly, some tasks involve only local resources (e.g. those in a single switch), whereas others might involve resources across many networks (e.g. optimizing the position of a videoconference server in a multicountry conference call). Finally, some tasks require detailed network knowledge (e.g. selecting the right QoS parameters for a guaranteed session), while others require domain specific knowledge (e.g. determining the computational cost of a MPEG to JPEG conversion). Given the diversity of these tasks, software engineering principles argue against building a single monolithic, complex resource management mechanism. Instead, the Darwin architecture comprises a small family of related mechanisms:

- *High-level resource allocation*: this mechanism, sometimes called a resource or service broker, performs global allocation of the resources based on a high-level application request, typically using domain knowledge for optimization. Its tasks include performing tradeoffs between services (e.g. trading computation for communication) according to the application-selected value metric (e.g. minimize cost, maximize service quality). 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 also want to provide the ability in some cases to interconnect incompatible services or endpoints, for example automatically inserting a transcoder service between two otherwise incompatible videoconference participants. The broker in the Darwin system is called *Xena* [8].
- *Low-level resource allocation*: this mechanism is a signaling 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 signaling protocol is called *Beagle*[9].
- *Runtime resource management*: this mechanism injects application or service specific 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

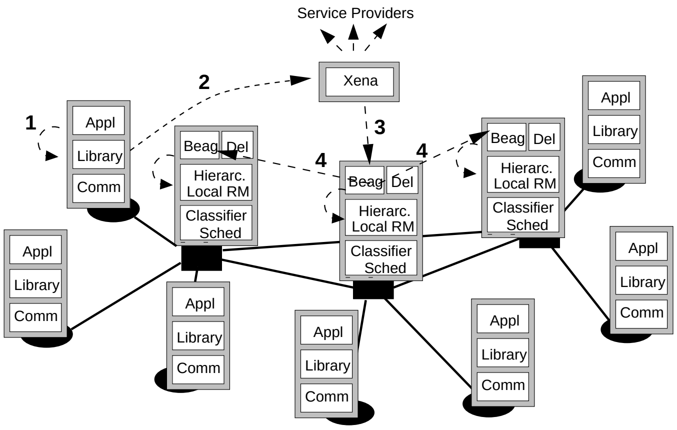

Figure 1. Darwin components

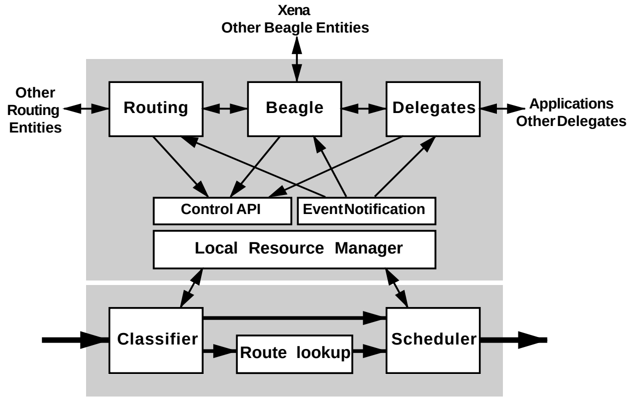

Figure 2. Darwin node software architecture

level service providers or organizations, 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.

These mechanisms were chosen because they cover the space, organization, and time requirements outlined in Section 2, and because they support resource management on a variety of time scales and scopes.

The different resource management mechanisms can be tied together using the abstraction of a *virtual network*. A virtual network, sometimes also called a virtual mesh [23] or supranet [12], is the set of resources that are allocated and managed in an integrated fashion to meet specific needs. Virtual networks can be used at the application and service provider level. An application mesh would be created in response to an application service request, while a service mesh captures the resources controlled by the provider to meet service requests. A virtual network not only captures resources, but it also can include state and code that represents or implements resource management policies that are appropriate for those resources. For example, the Darwin delegates that implement customized runtime resource management are associated with the virtual network. The creation of the virtual network provides an opportunity to do global resource optimization and to establish state in the virtual network that will speed up runtime adaptation.

3. Darwin

3.1. Darwin System Architecture

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 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 what 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 shows the components on each switch node and their most important interactions. The bottom part of the picture corresponds to the data plane. The focus in this component is on simplicity and high throughput. The top half 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. We expect that in general, the local resource manager will 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 IETF in the last few years rely on QoS routing [28, 16] (resource brokers), RSVP [32] (signaling similar to Beagle), and local resource managers that set up packet classifiers and schedulers. The more recent proposals for differentiated service [33] 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 not only communication, but also computation and storage resources.

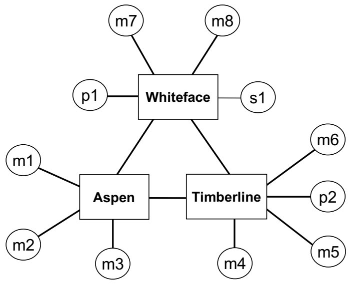

Figure 3. Darwin IP testbed topology

3.2. Darwin Testbed

The Darwin system has been implemented; and throughout this paper we show the results of a variety of experiments conducted on the Darwin network testbed. The topology of the testbed is shown in Figure 3. The three routers are Pentium II 266 MHz PCs running FreeBSD 2.2.5. The end systems `m1` through `m8` are Digital Alpha 21064A 300 MHz workstations running Digital Unix 4.0. The end systems `p1` and `p2` are Pentium Pro 200 Mhz PCs running NetBSD 1.2D and FreeBSD 2.2.5 respectively, and `s1` is a Sun Ultrasparc workstation running Solaris 2.5. All links except that between `s1` and `whiteface` are full-duplex point-to-point Ethernet links configurable as either 100 Mbps or 10 Mbps. The link to `s1` runs only at 10 Mbps.

In the remainder of the paper we describe Xena, Beagle, delegates, and hierarchical scheduling in more detail. Figure 4 shows the example that we will use throughout the paper. The top of the figure shows the input to Xena. The center part shows the information that Xena passes to Beagle, and the bottom section shows the information used to set up the local resource manager.

4. Xena: Resource Allocation in Space

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 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 the signaling protocol Beagle (Section 7).

4.1. Xena Design

The application provides 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).

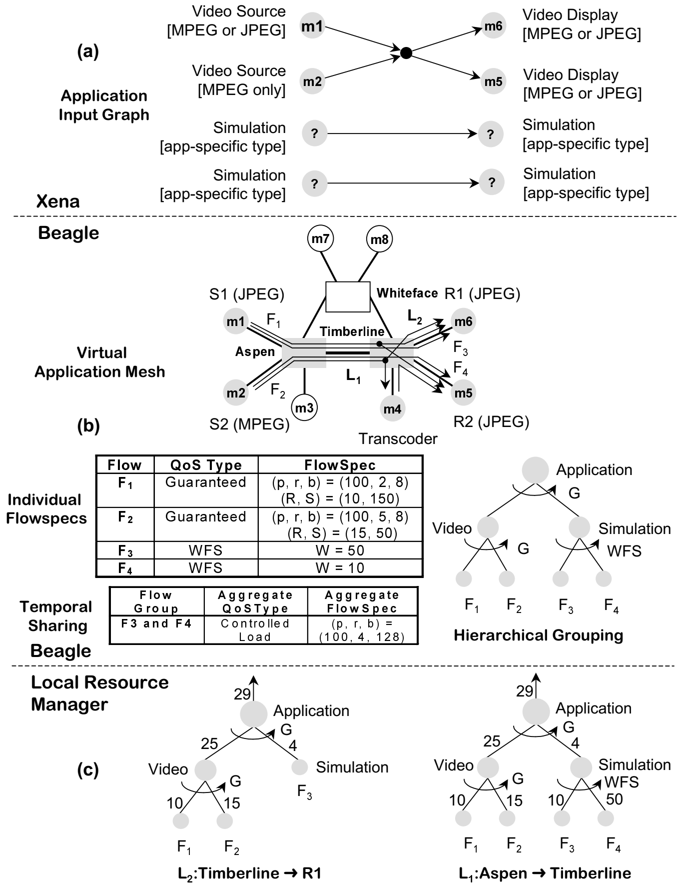

Figure 4. Handling an application service request in Darwin

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 matched pair of transcoders at two ends of the network segment. The first transcoder converts the flow to a more bandwidth efficient coding format (such as 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 that is 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, e.g. entities at flow endpoints must agree with the type of data that will 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 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. By using application specific criteria to guide resource selection, Xena in effect allows applications to customize the definition of service quality.

4.2. 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 request 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. 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 Mbps). Finally, there is a database that contains the various semantics-preserving transformations (e.g. JPEG-to-MPEG) and how to instantiate them.

Currently, Xena expresses its optimization problem in terms of an integer linear program, and turns it over to a solver package [5] that generates a sequence of successively better solutions at successively greater computation cost. Since the optimization problem is generally NP-hard, this approach is only appropriate for small to medium size problems. Work is in progress on defining heuristics and other simplifying assumptions that will make the problem tractable. This approach will necessarily trade quality for performance; i.e. our goal is to find high quality (but not, in general, optimal) solutions at a reasonable cost.

4.3. Example

Consider an application in which four scientists communicate via a videoconferencing tool that uses software MPEG/JPEG coders and collaborate on a distributed simulation that runs over an eight-node distributed computing testbed (depicted in Figure 4(b)).

Figure 4(a) shows the abstract resource mesh supplied to Xena. For the sake of clarity, some detail has been omitted; for example, we have depicted communication flowing in one direction only. For the video conferencing tool, since the scientists are physically located at their machines, the application provides to Xena specific network addresses (`m1`, `m2`, `m5`, `m6`) for the nodes participating in the videoconference. The nodes are also annotated by the requested service types (Video Source, Video Display). 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 flow's exact QoS parameters are left unspecified. For the distributed simulation, the application does not specify what nodes should participate.

The scientists request the *minimize cost* optimization strategy and at the time of the request, computation is costly relative to communication because of existing loads. This means that, even though it is simplest to use MPEG for the video, the less computation-intensive all-JPEG solution is more desirable (Figure 4). Xena can compensate for `m2`'s inability to emit JPEG by employing the MPEG-JPEG converter registered at `m4`; despite the detour through `m4` and its computation cost, this solution comes in at an overall lower cost than the all-MPEG approach. Moreover, due to severe loads on `m7`, `m8`, and the router `whiteface`, which are expressed as a high cost, the least costly solution collocates the distributed simulation nodes with the videoconference nodes, at `m1`, `m2`, `m5`, `m6`. Once node placement and route selection has occurred, Xena invokes Beagle to perform the resource allocation. The inputs to Beagle are shown in Figure 4(b); they are explained in more detail when we discuss signaling in Section 7.

5. Customizable Runtime Resource Management

We discuss how delegates can perform customized runtime resource management, describe the delegate runtime environment implemented in Darwin, and illustrate delegate operation using an example.

5.1. Delegates

We use the term "delegate" for code that is sent by applications or service providers to network nodes in order to

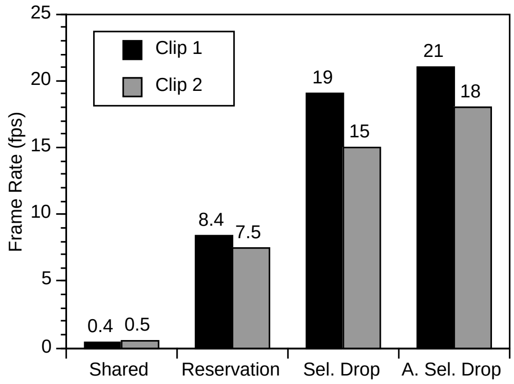

Figure 5. Video quality under four scenarios

implement customized management of their data flows. Delegates execute on designated routers and can monitor the network status and affect resource management on those routers through a control API, as shown in Figure 2. Delegates can be viewed as an application of active networks [25]: network users can add functionality to the network. It is, however, a very focused application. Delegate operations are restricted to traffic management, so delegates want to learn about changes in resource availability and must be able to change resource use.

A critical design decision for delegates is the definition of the control interface, i.e. the API that delegates use to interact with the environment. If the API is too restrictive, it will limit the usefulness of delegates, while too much freedom can make the system less efficient. The definition of the API is driven by the need to support resource management and it includes functions in a number of categories:

- Delegates can monitor the network status, e.g. congestion conditions.
- Delegates can change how resources are distributed across flows: splitting and merging of flows, changing their resource allocation and sharing rules, controlling selective packet dropping.
- Delegates can send and receive messages, for example to coordinate activities with peers on other routers or to interact with the application on endpoints.
- Finally, delegates can affect routing, for example to reroute a flow inside the virtual network for load balancing reasons, or to direct a flow to a compute server that will perform data manipulations to, for example, add compression or encryption to a flow.

The use of delegates 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 run time mechanisms (e.g. building "sandbox" that restricts what the delegate can access) and compile time mechanisms (e.g. proof carrying code). A related issue is that of security. At setup time, the router has to make sure that the delegate is

Methods	Description
`add`	Add node in scheduler hierarchy
`del`	Delete node from scheduler hierarchy
`set`	Change param. on scheduler queue
`dsc_on`	Activate selective discard in classifier
`dsc_off`	Deactivate selective discard in classifier
`probe`	Read scheduler queue state
`reqMonitor`	Request async. cong. notification
`retrieve`	Retrieve scheduler hierarchy

Table 1. Methods available at the Delegate/Local Resource Manager API

being provided by a legitimate user, and at runtime, the local resource manager has to make sure that the delegate only acts on flows that it is allowed to manage. The authentication and authorization of delegates will be performed jointly by the signaling protocol and the local resource manager; authentication and authorization have not yet been implemented.

5.2. Implementation

Our current framework for delegates is based on Java and uses the Kaffe Java virtual machine [29], capable of just-in-time (JIT) compilation and available for many platforms. 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." Currently, delegates can run with different static priority levels, although a more controlled environment with real-time execution guarantees is desirable.

The API that gives users access to resource management functions and event notification is implemented as a set of native methods that call the local resource manager, which runs in the kernel. Table 1 presents the methods of the Java class `delegate.Clschd`, which implements the API to the packet classifier and scheduler. Communication was built on top of standard `java.net` classes. Rerouting functions have not been implemented yet. While this environment is sufficient for experimentation, it is not complete. It needs support for authentication and mechanisms to monitor and limit the amount of resources used by delegates. We also expect the API to evolve over time.

5.3. Experiment

We present an example of how delegates can be used to do customized runtime resource management.

Suppose a network carries two types of flows, data and MPEG video, similar to the example in Figure 4. If packets are dropped randomly during congestion, the quality of the video degrades very quickly. The reason is that MPEG streams have three types of frames of different importance: I frames (intracoded) are self contained, P frames (predictive) uses a previous I or P frame for motion compensation and thus depend on this previous frame, and B frames (bidirectional-predictive) use (and thus depend on) previous and subsequent I or P frames. Because of these inter-frame dependencies, losing I frames is extremely damaging, while B frames are the least critical.

We direct three flows over the Aspen-Timberline link of the testbed: 2 MPEG video streams and an unconstrained UDP stream. Both video sources send at a rate of 30 frames/second, and our performance metric is the rate of correctly received frames. Figure 5 compares the performance of four scenarios. In the first scenario, the video and data packets are treated the same, and the random packet losses result in a very low frame rate. In the second case, the video streams share a bandwidth reservation equal to the sum of the average video bandwidths. This improves performance, but the video streams are bursty and the random packet loss during peak transfers means that less than a third of the frames are received correctly. In the third scenario, we also place a delegate on Aspen. The delegate monitors the length of queue used by video streams, and if the queue grows beyond a threshold, it instructs the packet classifier to identify and drop B frames; B frames are marked with an application-specific identifier. Packet dropping stops when the queue size drops below a second threshold. Figure 5 shows that is quite effective: the frame rate roughly doubles. The reason is that by restricting the presence of B frames in the queue, the I and P frames are protected from corruption.

While delegates provide an elegant way of selectively dropping B frames, the same effect could be achieved by associating different priorities with different frame types, i.e. layered video coding. In scenario four we use a delegate to implement a more sophisticated customized drop policy. In scenario three, typically too many B frames are dropped, because all flows are simultaneously affected. A better approach is to only drop the B frames of a subset of the video streams, assuming that is sufficient to relieve congestion. The advantage of having a delegate control selective packet dropping is that it can implement customized policies for controlling what video streams are degraded. Scenario four in Figure 5 shows the results for a simple "time sharing" policy, where every few seconds the delegate switches the stream that has B frames dropped. This improves performance by another 10-20%. Policies that differentiate between flows could similarly be implemented.

6. Hierarchical Scheduling

For an 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. Figure 4(c) illustrate a resource subtree for an application.

The ability to customize resource management policies at all sharing levels for a resource is one of the key requirements and distinctive features for 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. This section describes the Hierarchical Fair Service Curve (H-FSC) scheduling algorithm (first described in [24]) which meets the above constraint.

6.1. H-FSC Algorithm and Customization Features

In a H-FSC scheduler, associated with each link is a class hierarchy that specifies the resource management policy. Each interior class represents some aggregate of traffic flows that are managed by an entity such as the link owner, a service provider, and so on. The resource management policy for each entity is then mapped to one that can be implemented by the Fair Service Curve (FSC) algorithm. The goal of the H-FSC algorithm is to simultaneously satisfy all the FSC policies of all entities in the hierarchy.

In FSC, a stream i is said to be guaranteed a service curve $S_i(\cdot)$, if for any time t_2, there exists a time $t_1 < t_2$, which is the beginning one of stream i's backlogged periods (not necessarily including t_2), such that the following holds

$$S_i(t_2 - t_1) \leq w_i(t_1, t_2), \tag{1}$$

where $w_i(t_1, t_2)$ is the amount of service received by session i during the time interval $(t_1, t_2]$.

Consider an entity that manages N streams, where each stream can be an application flow, or a flow aggregate. The amount of resource the entity manages is the service guaranteed by its parent entity, denoted by $S(t)$. Without going into the details of the FSC algorithm, we state that the FSC algorithm can (1) guarantee the service curves for all streams if the stability condition $\sum_{i=1}^{N} S_i(t) \leq S(t)$ holds; (2) fairly allocate the excess service if some flows cannot use their guaranteed service.

We decided to use the H-FSC scheduler in Darwin because of the following two important properties of H-FSC. First, as long as the stability condition holds, H-FSC allows FSC policies to be combined arbitrarily in a hierarchy while satisfying all of their requirements simultaneously. Second, the FSC algorithm, which is used at each level in a H-FSC scheduler, provides a general framework for implementing many policies. For example, by ensuring a minimum asymptotic slope of $S_i(t)$ independent of the number of competing streams, a guaranteed bandwidth service is provided to stream i. By assigning $S_i(t)$ to be $S(t) \cdot \phi_i / \sum_{j=1}^{N} \phi_j$ for all streams, a weighted fair service in which stream i has a weight of ϕ_i is implemented. Unlike various fair queueing algorithms, FSC decouples delay and bandwidth allocation.

Due to the flexibility of FSC schedulers in implementing general policies and the flexiblity of H-FSC in arbitrarily integrating various FSC policies, various entities (resource own-

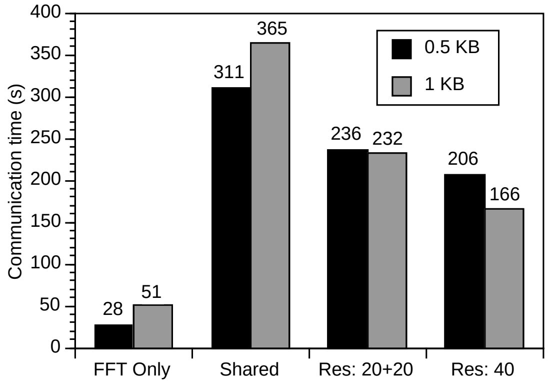

Figure 6. FFT communication times under various scenarios.

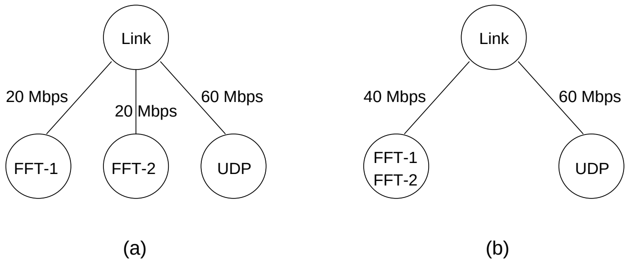

Figure 7. Resource trees in FFT experiments. (a) Per-flow reservation. (b) Aggregate reservation.

ers, service providers, applications) sharing the resource at different levels can *independently* customize or specify the policies of managing their share of the virtual resources.

6.2. Implementation

The implementation of the H-FSC scheduler in Darwin requires two extra components: a packet classifier and an API for signaling protocols.

Before they can be scheduled, incoming data packets must be classified and mapped to packet queues. To support traffic flows of various granularities, we have devised a highly flexible classification scheme. Each flow is described by a 9 parameter *flow descriptor*: source IP address, CIDR-style source address mask, destination IP address, CIDR-style destination address mask, protocol number, source port number, destination port number, application specific ID (carried as an option in the IP header), and CIDR-style application specific ID mask. A zero denotes a "don't care" parameter. Using this scheme, flows such as end-to-end TCP connections, aggregates of traffic between networks, and WWW, FTP, TELNET services can be specified. With the application specific ID, we can even sub-divide an end-to-end traffic flow into application specific sub-flows, such as the different frame types in a MPEG flow. The control API exported by the scheduler is discussed in Section 5. The processing overhead associated with classification and queueing in our implementation is suitably low for use in our Darwin testbed. Classification overhead is 3 μs per packet with caching on a 200 MHz Pentium Pro system. Average queueing overhead is around 9 μs when there are 1000 flows in the system. This low overhead allows us to easily support link speed of 100 Mbps in our prototype network testbed.

6.3. Experiments

We present the results of a set of experiments that demonstrate the importance of being able to reserve resources and to control dynamic bandwidth sharing.

In the first set of experiments, we run two distributed computations, Fast Fourier Transforms (FFTs), simultaneously to show the effect of reserving resources. Hosts `m1`, `m4`, and `m7` participate in the first mesh (FFT-1) on 0.5 KB of data for 96 iterations. Hosts `m2`, `m5` and `m8` participate in the second FFT mesh (FFT-2) on 1 KB of data for 32 iterations (please refer to Figure 3 for the testbed topology). All network connections in this set of experiments are configured at 100 Mbps. The all-to-all communication in the FFT often dominates the execution time. FFT uses TCP for communication and the communication pattern is highly bursty: the burst data rate is approximately 80 Mbps. Figure 6 summarizes the results. Scenario 1 shows the base case: in an idle network and without reservations, the communication times are 27.66 s and 51.05 s for FFT-1 and FFT-2 respectively.

In Scenario 2, we introduce two 90 Mbps UDP traffic streams from `m3` to `p1` and from `m6` to `p1`, causing congestion on the inter-router links. Since the FFTs use TCP for communication, their performance degrades significantly because the competing traffic does not use congestion control. To improve the FFTs' performance, protection boundaries must be drawn between the background traffic and the FFTs. In Scenario 3, we use reservations to guarantee that each FFT mesh gets at least 20 Mbs of bandwdith on each inter-router; Figure 7(a) shows the corresponding resource tree. The remaining bandwidth is given to the background traffic. Under this per-flow reservation scheme, the communication times of the FFTs improved by 24% and 36%. In Scenario 4, we apply a shared reservation of 40 Mbps for both FFTs; Figure 7(b) shows the resource tree. Since the FFT traffic is very bursty, the advantage of dynamically sharing the reserved bandwidth is significant: the communication time is reduced by another 13% and 29%. We conclude that it is important to be able to reserve resources, not only for flows, but also for flow aggregates.

In our second set of experiments, we demonstrate how hierarchical scheduling can be used to control dynamic bandwidth sharing. Consider a distributed interactive simulation application that combines an FFT with an interactive componenent such as a video stream or shared white-board. In this experiment, the FFT (1 KB of data and 32 iterations) uses `m2`, `m5`, and `m8`, while the interactive adaptive component is modeled by a TCP connection from `p2` to `s1`. In addition, background traffic is modeled by a full-blast UDP traffic stream from `m6` to `p1`. With "vanilla" resource distribution, we reserve 40 Mbps for the FFT mesh, 1 Mbps for the TCP stream, and the

	Non-hierarchical scheduling	Hierarchical scheduling
FFT comm. time (s)	70.76	73.18
TCP BW (Mbps)	1.41	5.30

Table 2. Scheduling performance comparison.

remaining bandwidth is given to the background UDP stream (Figure 8(a)). With this reservation scheme, the TCP stream achieves a throughput of 1.41 Mbps, while the FFT communication time is 70.76 s (Table 2).

With hierarchical resource management we group flows as is shown in Figure 8(b). While the overall reservation for the application remains the same, the hierarchy specifies that TCP (FFT) gets the first chance at using any bandwidth left unused by FFT (TCP). The result is that the throughput of the TCP stream is now 5.30 Mbps, a **276%** improvement, while the communication time of the FFT increases by an insignificant amount of 3% to 73.18 s (Table 2). This example demonstrates that hierarchical scheduling makes it possible for applications to cooperatively share the reserved resources more effectively within the application boundary and that specifying specific resource trees, applications and service providers can customize how resources are shared.

7. Beagle: Signaling

The Beagle signaling protocol provides support for the customizable resource allocation model of Darwin. While traditional signaling protocols such as RSVP [32] and PNNI [3] which operate on individual flows, Beagle operates on virtual networks or meshes. We elaborate on some key Beagle features in this section — more details can be found in [9].

7.1. Beagle Design

On the input side, Beagle interfaces with Xena to obtain the virtual network specification generated by Xena. The virtual network is described as a list of flows and delegates, plus resource sharing specifications that describe how flows within a mesh share resources amongst them. The example virtual network in Figure 4(b) shows two video flows and two distributed interactive simulation flows. Each flow is specified by a *flow descriptor* as described in Section 6.2 and information such as a tspec and a flowspec object, as in the IETF IntServ working group model. A delegate is characterized by its resource requirements (on CPU, memory, and storage), its runtime environment, and a list of flows that 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, etc.). The virtual network typically also includes a number of designated routers which identify the mesh core. In the example, Aspen and Timberline are the designated routers.

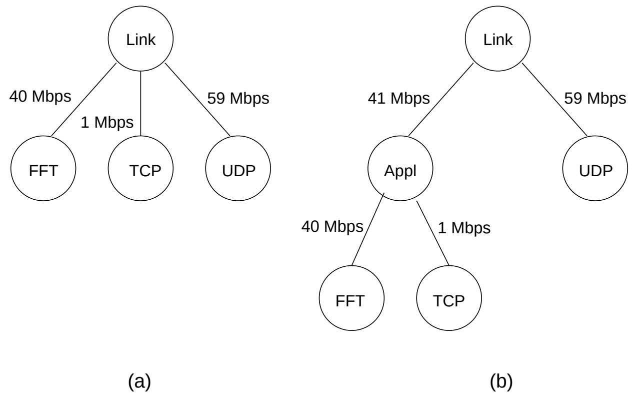

Figure 8. Resource trees in distributed interactive simulation experiments. (a) Per-flow reservation. (b) Hierarchical reservation.

After receiving a request, Beagle issues a sequence of flow setup messages to the different nodes in the mesh, each message specifying what total resources are needed on a link, plus a grouping tree that specifies how these resources should be applied. An efficient setup of a mesh includes the establishment of the core, and individual flow setups initiated by the senders or receivers that rendezvous with the core in designated routers. Our initial implementation uses simple per-flow setups and a hard state approach based on three-way handshakes between adjacent routers. Future work includes optimizing mesh setup and evaluating the use of soft state for some or all of the mesh state.

On each node, Beagle passes a resource trees (Figure 4(c)) to the Local Resource Manager to allocate resources for flows. This interface is similar to the one described in Section 5 (Table 1). Beagle also establishes delegates onto switch nodes (for resource management delegates), or compute 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. In the future, Beagle will also provide support for communication channels between control delegates belonging to the same service and also provide mechanisms to recover from control delegate failures.

7.2. Resource Distribution

In Section 6 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 physical 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. Figure 4(b) gives an example.

The hierarchical grouping tree encodes the hierarchical sharing structure of all flows sharing a virtual link. Once it knows the actual flows that share a particular physical link in the network, Beagle prunes the hierarchical grouping tree, eliminating flows that 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 pruning the grouping tree to eliminate flows that do not exist at a particular link and converting flowspecs at each node into appropriate low-level scheduler-specific parameters, such as a weight for hierarchical weighted fair share schedulers [4] or a service curve for the Hierarchical Fair Service Curve scheduler [24].

7.3. Temporal Sharing

There are often resource sharing opportunities on time scales larger than what 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 like to associate an aggregate bandwidth requirement for a group of best-effort flows. 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]. However, control of flow routing using designated routers allows us to take better advantage of sharing opportunities. The temporal sharing object also generalizes RSVP's notion of resource reservation styles. However, RSVP limits aggregation to flows within a multicast session and the aggregate flowspecs must be the result of either a sum or a least upper bound (LUB) operation on the individual flowspecs. In Beagle, the temporal sharing object can be used to group arbitrary flows within an application mesh and any aggregate flowspec can be used. As an example, in Figure 4(b), the distributed interactive simulation application associates an aggregate Controlled Load service flowspec with the two simulation flows.

The hierarchical grouping tree and the temporal sharing objects both define ways in which an application can tailor resource allocation within the mesh. However, they are separate concepts and are orthogonal to each other. If both types of sharing are specified, the resource tree is derived by applying

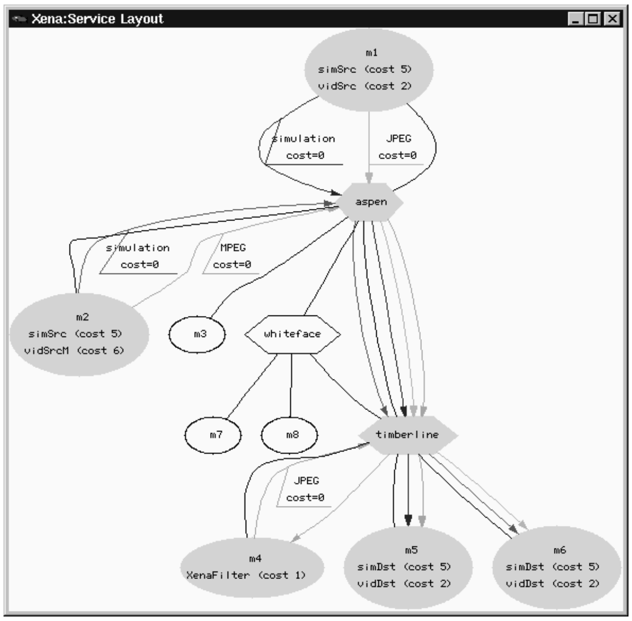

Figure 9. Xena output showing service layout

the temporal sharing specification at every level of the tree. If the temporal sharing object lists flow groups that do not fall under the same parent node in the resource tree, the temporal sharing behavior is ignored. The example in Figure 4(b) shows the use of both sharing objects. The resulting link resource sub-trees at links L_1 and L_2, assuming the use of hierarchical weighted fair share schedulers [4], are shown in Figure 4(c).

7.4. Experimental Evaluation

To evaluate the performance of the Beagle prototype implementation, we measured end-to-end setup latencies for flows and delegates, and per-router flow setup processing times on the Darwin testbed. The experiment involved setting up the virtual network shown in Figure 4(b). All measurements reported are averages from 100 runs. The average end-to-end latency through two routers was 7.5ms for flow setup and 3.8ms for delegate setup. The flow setup processing time on each router was 2.4ms, about 68% of which was spent in interacting with the local resource manager. This involves admission control and setting up flow state in the packet classifier and scheduler. The current Beagle prototype supports about 425 flow setups per second. This is comparable to connection setup times reported for various ATM switches. However, we expect improvement in these results by optimizing the implementation.

8. Darwin System Demonstration

This section demonstrates the various pieces of the Darwin system in action. The example used in demonstrating the system is the same as the one shown in Figure 4(b). The service layout produced by Xena is shown in Figure 9 which is a screenshot of a running Xena system. As shown in Figure 9,

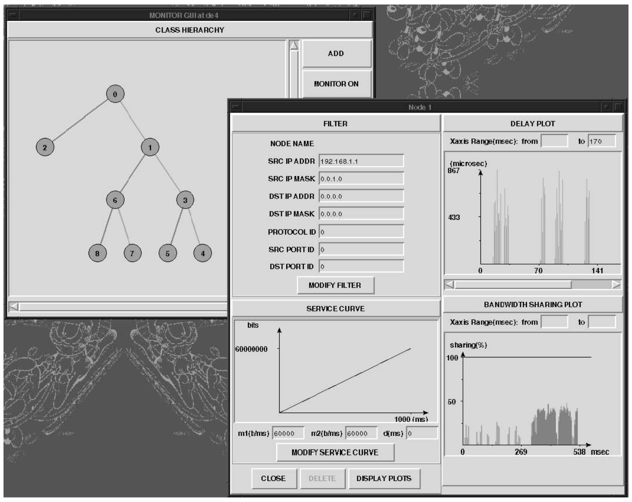

Figure 10. Scheduler graphical user interface showing the resource tree and bandwidth plots

Xena instantiates a transcoder delegate on m4 to convert the MPEG video flow to JPEG format. Xena also selects m1 and m2 as simulation endpoints. Beagle takes this service layout and allocates resources for the video and simulation flows on Timberline and Aspen. Figure 10 is a screenshot of a user-interface program for the H-FSC scheduler showing the classifier and scheduler state installed on link L_1 in the direction from Aspen to Timberline. Figure 10 also shows the bandwidth plots for the video (light grey) and simulation (dark grey) flows at that link.

9. Related Work

There has recently been a lot of work as part of the Xbind [19, 31] and TINA [13, 26] efforts to define a service-oriented architecture for telecommunication 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 paper 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 Xbind and TINA have so far primarily been used as a development framework for traditional ATM and telecommunication network management mechanisms, they could potentially also be used as a basis for the development of customizable resource management mechanisms.

Over the past decade much work has gone into defining QoS models and designing associated resource management mechanisms for both ATM and IP networks [10, 14, 27]. This has resulted in specific QoS service models both for ATM [1] and IP [7, 30, 22]. This has also resulted in the development of QoS routing protocols [3, 28, 21, 20] and signaling protocols [2, 3, 32]. A closely related issue being investigated in the IP community is link sharing [15], 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 communcation services, Darwin addresses both bitway and value-added service providers. Second, most QoS models only support QoS on a per-flow basis [7, 1]. Exceptions are the concept of VP and VC in ATM, and IP differential service model [6, 18, 11, 33], but these efforts are very restricted either in the type of hierarchy they support or in the number of traffic aggregates for which QoS can be provided. In contrast, Darwin uses virtual networks 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 towards resource management along these three dimensions.

The idea of "active networks" has recently attracted a lot of attention. In an active network, packets carry code that can change the behavior of the network [25]. 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. Second, Darwin's facilities for managing both computation and communication resources via virtual networks can help to solve key resource allocation problems faced by active networks.

10. Summary

We have designed a resource management system called Darwin for service-oriented networks that takes an integrated view towards resource management along space, time, and organization dimensions. The Darwin system consists of four inter-related resource management mechanisms: resource brokers called Xena, signaling protocol called Beagle, runtime resource management using delegates, and hierarchical scheduling algorithms based on service curves. The key property of all these mechanisms is that they can be customized according to service specific needs. While these mechanisms are most effective when they work together in Darwin, each mechanism can also be used in a plug-and-play fashion in 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 paper implements the vision and demonstrates some of the possibilities, but much work remains to be done. Future versions will be far more scalable, both in terms of routing in large topologies and in terms of 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 the current project: tools for service creation, mechanisms for automated discovery of resources, and detailed accounting of resource commitment and use.

References

[1] ATM Forum Traffic Management Specification Version 4.0, October 1995. ATM Forum/95-0013R8.

[2] ATM User-Network Interface Specification. Version 4.0, 1996. ATM Forum document.

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

[4] J. Bennett and H. Zhang. Hierarchical packet fair queueing algorithms. In *Proceedings of the SIGCOMM '96 Symposium on Communications Architectures and Protocols*, pages 143–156, Stanford, August 1996. ACM.

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

[6] S. Blake. Some issues and applications of packet marking for differentiated services, July 1997. Internet Draft.

[7] R. Braden, D. Clark, and S. Shenker. Integrated services in the internet architecture: an overview, June 1994. Internet RFC 1633.

[8] P. Chandra, A. Fisher, C. Kosak, , and P. Steenkiste. Network support for application-oriented qos. In *Proceedings of Sixth International Workshop on Quality of Service*, pages 187–195, Napa, California, USA, May 1998. IEEE.

[9] P. Chandra, A. Fisher, and P. Steenkiste. Beagle: A resource allocation protocol for an application-aware internet. Technical Report CMU-CS-98-150, Carnegie Mellon University, August 1998.

[10] D. Clark, S. Shenker, and L. Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. In *Proceedings of ACM SIGCOMM'92*, pages 14–26, Baltimore, Maryland, Aug. 1992.

[11] D. Clark and J. Wroclawski. An approach to service allocation in the internet, July 1997. Internet draft, draft-clark-diff-svc-alloc-00.txt, work in progress.

[12] L. Delgrossi and D. Ferrari. A virtual network service for integrated-services internetworks. In *Proceedings of the 7th International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 307–311, St. Louis, May 1997.

[13] F. Dupuy, C. Nilsson, and Y. Inoue. The tina consortium: Toward networking telecommunications information services. *IEEE Communications Magazine*, 33(11):78–83, November 1995.

[14] D. Ferrari and D. Verma. A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communications*, 8(3):368–379, Apr. 1990.

[15] S. Floyd and V. Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4), Aug. 1995.

[16] R. Guerin, D. Williams, T. Przygienda, S. Kamat, and A. Orda. QoS Routing Mechanisms and OSPF Extensions. *IETF Internet Draft <draft-guerin-qos-routing-ospf-03.txt>*, March 1998. Work in progress.

[17] A. Gupta, W. Howe, M. Moran, and Q. Nguyen. Resource sharing in multi-party realtime communication. In *Proceedings of INFOCOM 95*, Boston, MA, Apr. 1995.

[18] K. Kilkki. Simple integrated media access (sima), June 1997. Internet Draft.

[19] A. Lazar, K.-S. 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.

[20] Q. Ma and P. Steenkiste. On path selection for traffic with bandwidth guarantees. In *Fifth IEEE International Conference on Network Protocols*, pages 191–202, Atlanta, October 1997. IEEE.

[21] Q. Ma and P. Steenkiste. Quality of service routing for traffic with performance guarantees. In *IFIP International Workshop on Quality of Service*, pages 115–126, New York, May 1997. IFIP.

[22] S. Shenker, C. Partridge, and R. Guerin. Specification of guaranteed quality of service, September 1997. IETF RFC 2212.

[23] P. Steenkiste, A. Fisher, and H. Zhang. Darwin: Resource management in application-aware networks. Technical Report CMU-CS-97-195, Carnegie Mellon University, December 1997.

[24] I. Stoica, H. Zhang, and T. S. E. 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.

[25] D. Tennenhouse and D. Wetherall. Towards and active network architecture. *Computer Communication Review*, 26(2):5–18, April 1996.

[26] Tina consortium. http://www.tinac.com.

[27] J. S. Turner. New directions in communications (or which way to the information age?). *IEEE Communications Magazine*, 24(10):8–15, October 1986.

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

[29] T. Wilkinson. KAFFE - A virtual machine to run Java code. http://www.kaffe.org/.

[30] J. Wroclawski. Specification of the controlled-load network element service, September 1997. IETF RFC 2211.

[31] Project X-Bind. http://comet.ctr.columbia.edu/xbind.

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

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