Beagle: A Resource Allocation Protocol for Advanced Services Internet

Prashant Chandra Allan Fisher Peter Steenkiste

October 1998

CMU-CS-98-150

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

There is an emerging class of multi-party, multimedia, multi-flow applications (e.g distributed, interactive simulation) that have a high-level structure that imposes dependencies between resource allocations for flows within the application. These applications are also capable of making intelligent decisions on how resource allocation should be controlled within the application. The development of such applications is enabled by an Internet that is evolving towards providing ubiquitous, value-added, customizable services. Resource management mechanisms can exploit the high-level structure of applications to achieve better quality for applications and improved resource efficiency for the network. This paper describes a resource allocation protocol that allows this class of applications to optimize resource allocation using high-level application-domain knowledge.
The *Beagle* protocol uses an *application mesh* as the basic unit of resource allocation. The application mesh is a logical entity that encapsulates all the computation and communication resources that are allocated to an application at end-points and inside the network. Beagle provides support for allocating computational resources and for *delegates* — downloadable code fragments that can be used to customize resource management during runtime. Beagle provides a way for applications to optimize resource allocation by expressing a wide range of sharing policies to share resources amongst its flows. Beagle also provides support for aggregation of resource allocation at various granularities. This paper discusses the design of the Beagle components, describes a prototype implementation and presents preliminary performance evaluation results.

This research was sponsored by the Defense Advanced Research Projects Agency and monitored by NCCOSC RDTE Division, San Diego, CA 92152-5000, under contract N66001-96-C-8528.
The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. government.

Keywords: Computer networks, Resource allocation, Signaling protocols

1 Introduction

Until the early 1990s, the Internet only provided a best-effort, point-to-point datagram service in the network layer. This service was sufficient for simple applications such as telnet and ftp which were predominant at that time. Resource management mechanisms in the Internet control plane were restricted to dynamic re-routing around failures. The introduction of multicast [1] in the early 1990s and the subsequent development of audio-visual applications like vic, vat, etc., introduced a loose notion of "application flows" into the Internet control plane in the form of multicast distribution trees. Over the past few years, the Internet community has focussed on developing an Integrated Services Internet architecture [2], which is capable of delivering Quality of Service (QoS) to individual application flows. This has led to the development of control plane protocols such as SCMP [3] and RSVP [4] that can allocate resources to flows. These protocols enable the development of real-time, multimedia applications on the Integrated Services Internet.

While the development of a QoS-capable Internet and resource allocation protocols to support multi-party, multimedia applications represents a significant development, the Internet continues to evolve towards a global infrastructure capable of delivering a wide range of services. These services are *value-added* in the sense that they can possibly modify the data as it flows through the network; and *customizable* in the sense that their behavior can be tailored to meet specific needs of individual applications. Such an *advanced services Internet* enables the development of applications which in addition to communication resources, can also use computation and storage resources available in the network. Examples of services that can be provided on an advanced services Internet are reliable multicast, intelligent caching, video/audio transcoding, mixing and real-time language translation.

Resource management is one of the key components of the overall service-oriented network architecture. Resource management mechanisms for an advanced-services Internet, must be able to take an integrated view of the application as a whole, while providing maximum freedom for applications to customize the distribution of resources to its flows. This allows applications to use high-level knowledge to achieve improved quality and resource efficiency. This paper describes the design of the Beagle resource allocation protocol, intended for structured applications developed for an advanced services Internet. Beagle allows applications to customize resource allocation during setup and resource management during runtime. Beagle also allows applications to allocate computational and storage resources in the network. Throughout the paper the terms "signaling protocol" and "resource allocation protocol" are used interchangeably. Though, in the classical sense, signaling indicates both connectivity and resource allocation, the distinction is blurred in the context of the Internet where best-effort connectivity is always available.

The rest of the paper is organized as follows. Section 2 outlines the motivation for this work. Section 3 describes an overall resource management model for advanced services networks. Section 4 describes the architecture of Beagle. A prototype implementation and preliminary performance evaluation results are presented in Section 5. Section 6 contrasts Beagle with related work and Section 7 presents the conclusions.

2 Motivation

Current resource allocation protocols for integrated services networks use flows as the basic unit of resource allocation. Flows can be end-to-end application flows at the lowest granularity or can be aggregated virtual links leased by an organization. These protocols work well for simple applications with a single or few flows. However there is an emerging class of complex multimedia, multi-party applications which use several flows of various data rates and QoS requirements. These applications have a high-level structure that imposes dependencies between resource allocations for flows within the application. Moreover, these applications are capable of making intelligent decisions on how resource allocation should be controlled within the application. Current flow-based resource allocation protocols such as RSVP and PNNI [5] cannot make good use of the application structure information.

As a motivating example, consider a distributed interactive simulation application. This application has a many-to-many communication pattern with bandwidth requirements on both video and voice delivery to the participants and on data delivery between simulation end-points.

This example application places several new requirements on signaling protocols:

- *Topology customization* — the application might lease several compute servers as simulation nodes and place these servers at strategic points in the network with respect to the participants. The application can use application-domain knowledge to pick a compute server that either minimizes cost or maximizes performance. Other applications that have this requirement include distributed computer games, intelligent caching, etc.

- *Aggregation* — large distributed interactive simulation applications exchange data between several sites, with a large number of participating hosts per site. This requires aggregation of resource allocation requests from individual hosts into a single request between sites.

- *Flow manipulation* — the application may need to use generic transcoding services to perform video type matching amongst participants. It may also use video mixing, down sampling and other kinds of flow manipulation services. This requires allocation of computation and storage resources inside the network in addition to communication resources.

- *Resource sharing* — the application exhibits a high-level structure which defines how many participants can be active in the video conference at one time. The application may also want simulation flows to use bandwidth not used by bursty video flows. This high-level application domain knowledge can be used to have more efficient resource sharing within the application.

- *Resource optimization* — the application may want to route groups of multimedia flows together to promote better sharing and synchronization. Structured applications can optimize the allocation of resources amongst flows by controlling how flows are routed over the network. Other examples are using disjoint routes to achieve improved reliability and using multi path routes for best-effort flows to achieve improved throughput.

The requirements described above can be more easily realized with a resource allocation protocol that takes an integrated view of the application and can propagate and exploit its structure at appropriate times during resource allocation. Some advantages that result from having such a resource allocation protocol are: 1) improved resource efficiency results from cooperative sharing of resources among application flows, 2) more predictable behavior results from having an application manage its own resources during runtime, 3) semantic knowledge about application flows can be used to trade-off communication against computation, thereby enabling services that would otherwise not be possible due to resource unavailability. This paper describes a resource allocation protocol that is intended for this class of intelligent multimedia multi-party applications.

3 Structured Network Resource Management

In order to meet the requirements mentioned above, resource management mechanisms are needed that can incorporate knowledge about the application-domain in making resource allocation decisions at various levels. An example of such *structured network resource management* is the set of resource management mechanisms being developed by the CMU Darwin project [6]. As shown in Figure 1, applications running on end-points can submit requests for service to a resource broker (Xena) [7]. The resource broker invokes appropriate services, identifies the resources needed to satisfy the request and passes this information to a signaling protocol (Beagle) which allocates the resources. Some applications which include their own resource brokers, can interact directly with Beagle. At each node Beagle interacts with a local resource manager (LRM) to acquire and set up the resource.

The above architecture is driven in part by the goal of delivering customizable services in a heterogenous network environment, and by a need to support multiple administrative domains. Resource brokers interact directly with applications and can incorporate a lot of knowledge about the application domain. One example of a resource broker could be a service provider specializing in video conferencing: it can translate a high level request (high-quality video conference for 5 participants) into a list of required resources. A radically different example of a resource broker could be a network resource manager for a corporate virtual private network: the manager understands the relative priorities of requests and when additional resources can be leased.

While resource brokers are tightly coupled to applications, local resource managers are tightly coupled to the management of a specific communication, computation or storage resource. Several different types of LRMs may be deployed in different administrative domains. The Beagle resource allocation protocol is responsible for bridging the gap between applications/resource brokers and LRMs. Beagle supports the Darwin resource management model, by providing high-level abstract interfaces through which resource allocation and management can be customized by individual applications/resource brokers independent of the type of the LRMs on which the application runs.

Figure 2(a) shows the overall architecture of a Darwin switch/router node. Solid arrows indicate data plane interactions and dotted arrows indicate control plane interactions. The bottom half of the figure corresponds to the data plane. The focus in this component is on simplicity and high throughput. Data entering the node through one of the input interfaces is classified into one of several flows by the packet classifier. The classifier also provides the next hop

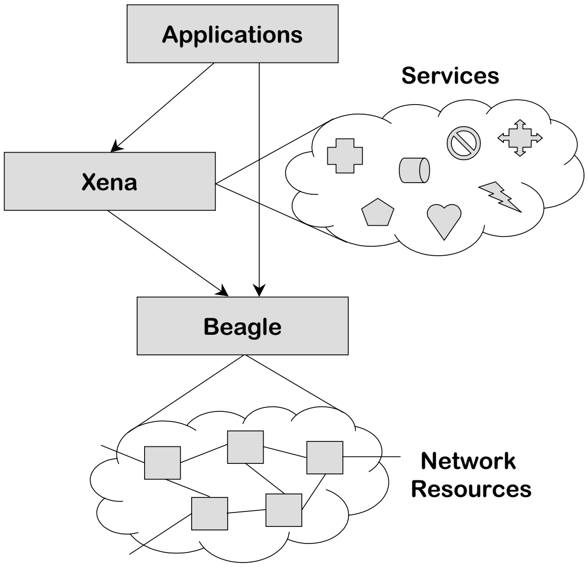


Figure 1: Components of the Darwin Architecture

information for the flow. At each output interface data is scheduled for transmit by a hierarchical link sharing scheduler. Activities in the control plane happen on a coarser time scale and provide customizable resource management to applications. The LRM exports a control API and an event notification interface. The control API is used by Beagle and *delegates* to setup and manage flow state in the classifier and scheduler. Delegates are application-specific code segments that execute in the interior of the network and provide customized resource management. Delegates will be discussed in more detail in Section 4.2. The event notification interface is used to deliver notifications about various events like failures, queue sizes exceeding a threshold etc., to Beagle and interested delegates. The control plane can run on a separate processor and control several data planes to achieve improved scaling.

Figure 2(b) shows the architecture of a Darwin end-system/compute server node. Here the data plane includes delegates which provide data manipulation functions such as transcoding, mixing, etc. The node also has a CPU scheduler which provides computational QoS to delegates. The flow classifier is responsible for delivering incoming flows to delegates for processing.

4 Beagle Architecture

This section describes the design of the Beagle resource allocation protocol. Throughout the section the example shown in Figure 3 will be used to illustrate the concepts.

The example is a distributed interactive simulation involving four collaborating scientists. A resource broker creates an application input to Beagle as shown in Figure 3(a). The input comprises of 1) several point-to-multipoint video flows providing video conferencing between the parties; 2) several point-to-point simulation flows exchanging data between the end-points (m1, m2, m5 and m6) and 3) a transcoder delegate responsible for changing the format of S2's video output from MPEG to Motion JPEG. For the sake of clarity, some detail has been omitted; for example, not all multicast branches of the full multipoint-to-multipoint communication are shown. Given this application and its resource requirements, Beagle has to 1) set up the application flows 2) set up the transcoder delegate and 3) realize the resource sharing between flows specified by the application. Each of these three actions is described in detail in the following sections.

4.1 Application Mesh

The application mesh is a logical entity that groups all the flows and delegates that comprise an application, for resource allocation purposes. The application mesh encapsulates all computation and communication resources allocated to the application both in the network and in end-systems. The application mesh also encapsulates policy information in

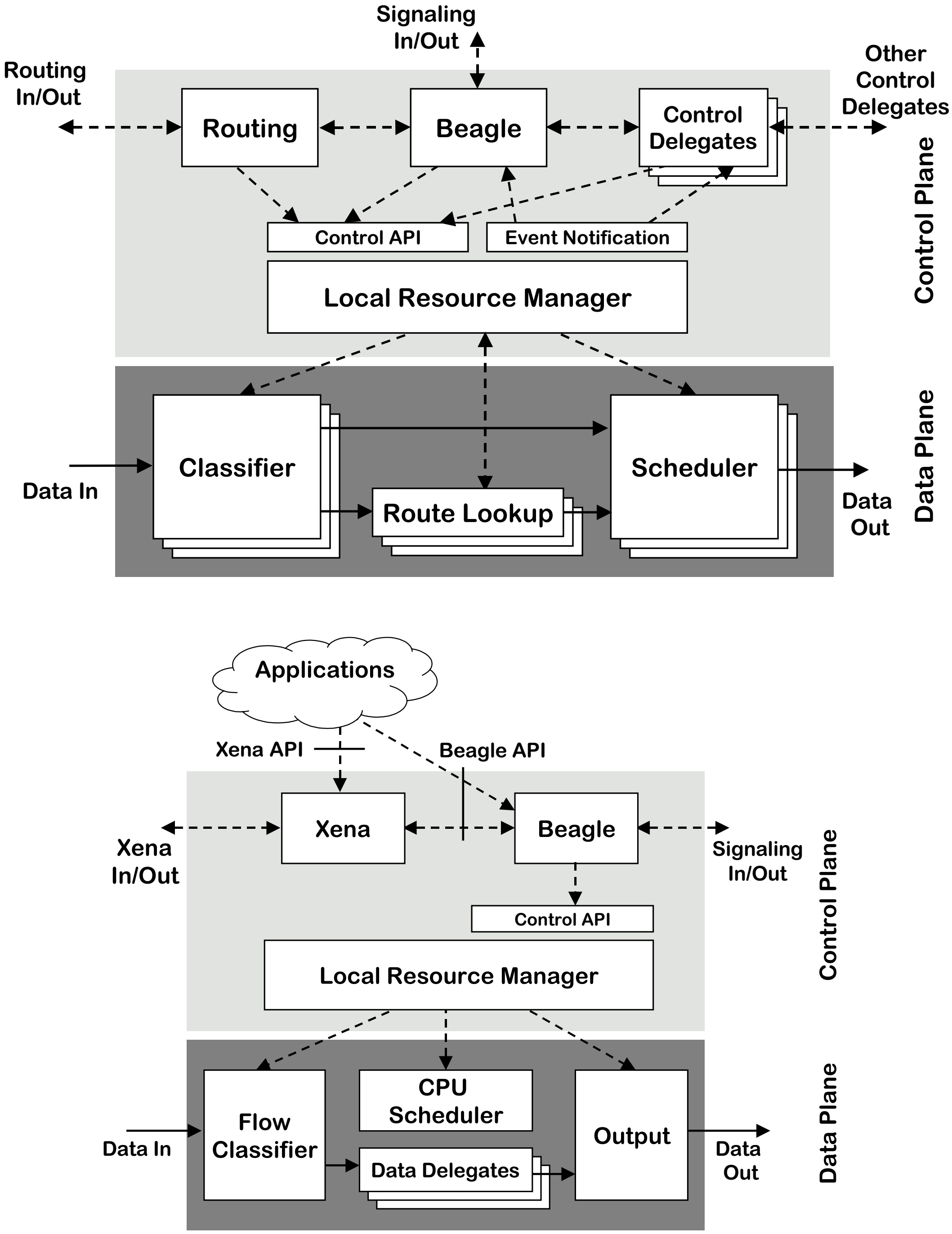
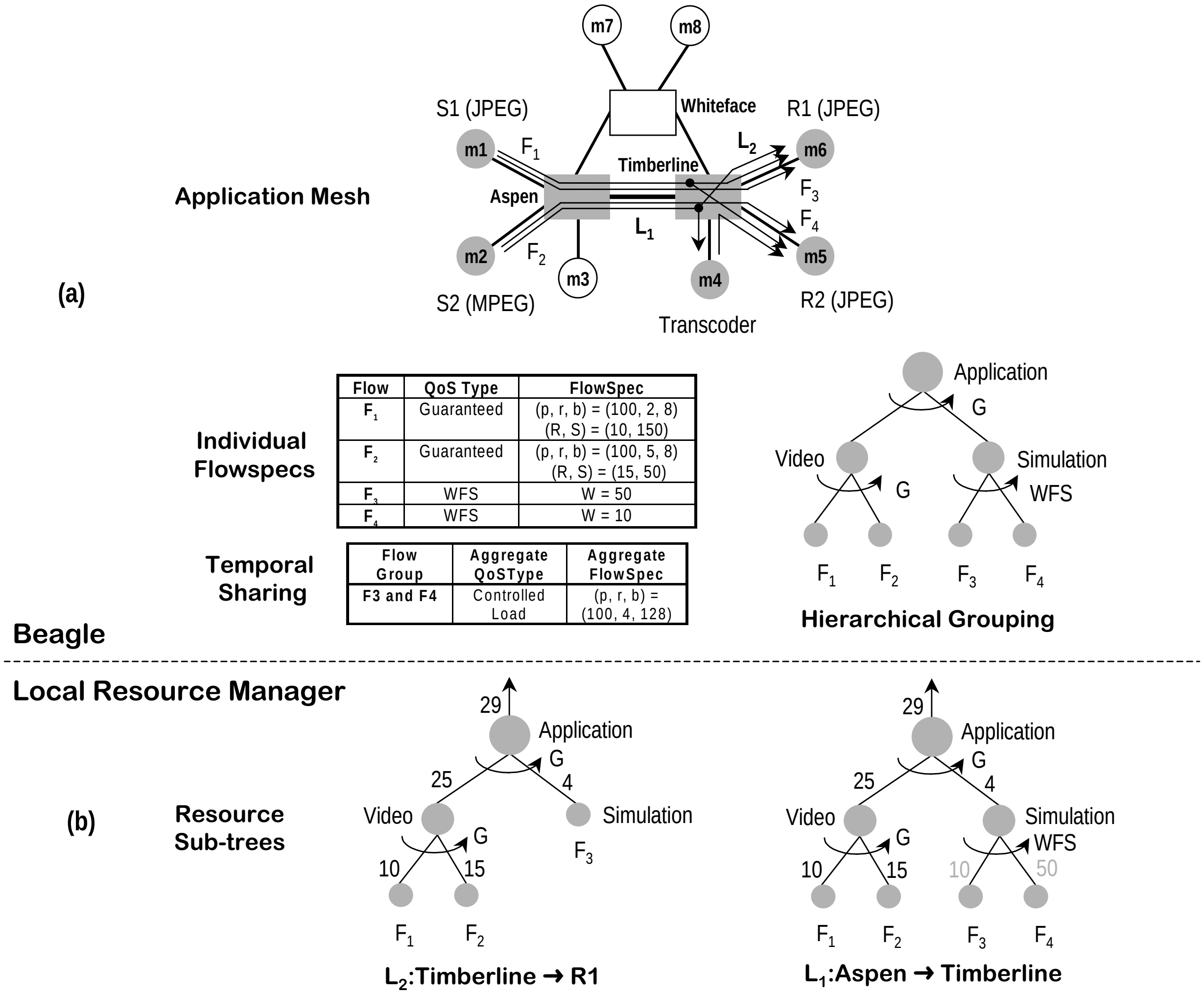


Figure 2: (a) Darwin switch/router node architecture. (b) Darwin end-system/computer server node architecture.

the form of resource sharing specifications that describe how flows within a mesh share resources amongst them. The mesh also includes a number of “designated” routers, with the requirement that certain flows pass through them. This is important for many reasons: for policy reasons, to take advantage of resource sharing by aggregation, or to direct a flow through a transcoder. Figure 3(a) shows the application mesh specification for the distributed interactive simulation application. Aspen and Timberline are the designated routers for this application. It is impractical for applications and resource brokers to have a picture of the network that identifies every link and node in the mesh at the lowest granularity. For this reason, links between designated routers in a mesh description are really *virtual links*.



Flow	QoS Type	FlowSpec
F_1	Guaranteed	(p, r, b) = (100, 2, 8) (R, S) = (10, 150)
F_2	Guaranteed	(p, r, b) = (100, 5, 8) (R, S) = (15, 50)
F_3	WFS	W = 50
F_4	WFS	W = 10

Flow Group	Aggregate QoSType	Aggregate FlowSpec
F3 and F4	Controlled Load	(p, r, b) = (100, 4, 128)

Figure 3: Handling an application request in Beagle.

A virtual link traverses several routers and is composed of many physical links. However, in the example shown, the virtual link between Aspen and Timberline is the same as the physical link L_1.

The application mesh is identified globally by an *application id object* which is a concatenation of the IP address of the owner node and a serial number allocated by the owner. Using this unique id, flows and delegates can be added/deleted from an application mesh during runtime. End-points can also be added/deleted from a given application mesh during runtime in a manner similar to IP multicast receiver joins and exits.

Beagle flows are defined by a globally unique *flow descriptor object*, which is a combination of the IP source and destination addresses, source and destination port numbers and an application-defined flow tag, carried as an IP option in the IP header. Application-defined flow tags are ignored by Beagle for resource allocation purposes, but are used by delegates to perform application-specific resource management tasks (e.g. selective discard of MPEG frames). Beagle flows can represent traffic at various granularites. End-to-end flows require a full-specification of all the flow descriptor fields. Aggregate flows are specified either by using a list of individual flow descriptors, or by using CIDR [8] style masks on source address, destination address and application flow tag fields, or by a combination of both. Each flow also has a *(beginIP, endIP)* address pair that denotes the “end-points” of the flow for resource allocation purposes. For end-to-end flows, beginIP and endIP are the same as source and destination addresses. For aggregate flows, beginIP and endIP represent the aggregation and de-aggregation points respectively. The QoS requirements of flows are expressed through the use of a sender tspec and a flowspec object — similar to the IETF IntServ working group model. In addition to Guaranteed Service [9] and Controlled Load Service [10], Darwin defines two new QoS

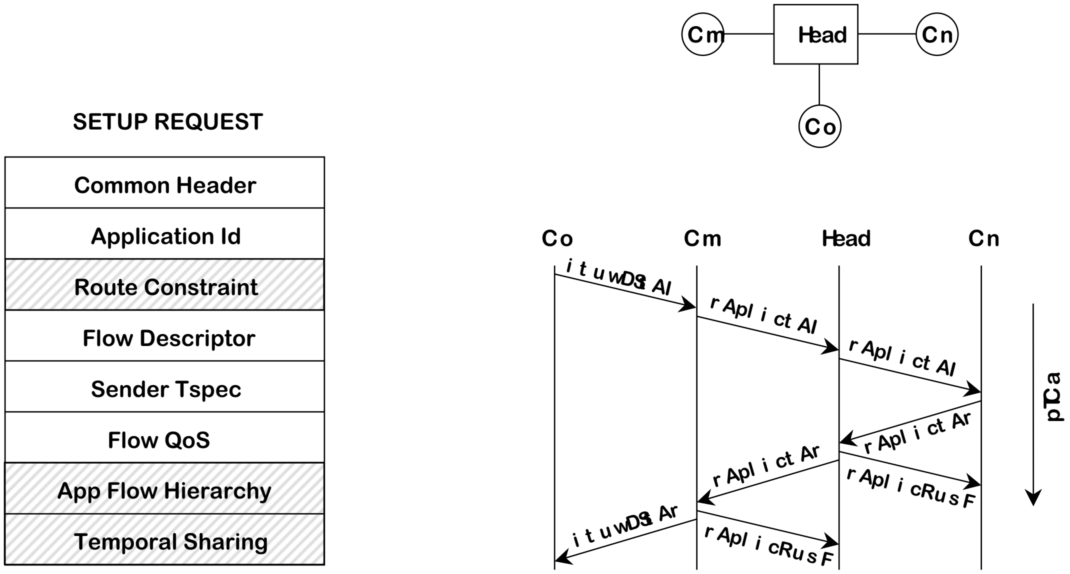


Figure 4: (a) Format of the SETUP REQUEST message, (b) Message flow for flow setup initiated by a third-party.

services — 1) Weighted Fair Share service, where flows share the available bandwidth in a weighted fair fashion and 2) Static Priority service, where flows are scheduled according to strict priority. As shown in Figure 3(a), the video flows use Guaranteed service, while the simulation flows use Weighted Fair Share service. Beagle treats all QoS specific objects as opaque objects which are interpreted only by the local resource managers. This makes it possible for Beagle to be extensible, and new QoS classes can be defined and added at any time.

Beagle flows can be unicast or multicast. A multicast flow is denoted by using a multicast group address in the destination address field. A multipoint-to-multipoint flow is composed of several multicast flows using the same group address. Each multicast flow originating from a source has an explicit set of receivers associated with it at setup time. Multicast flow setup proceeds along the data distribution tree from the source to the receiver set. Receivers can be dynamically added and deleted during runtime from various multicast flows without notifying the source. Beagle supports two modes of multicast: the first is a *structured mode* which has an explicit set of senders and receivers and the second is the more unstructured *IP multicast mode* which does not require senders to be members of the group and has no explicit receiver set. The structured mode is better suited for small to medium scale application meshes and the IP multicast mode works well for large uncoordinated meshes. There are several options for multicast data distribution: the mesh core can be used as a core-based tree, or explicit multicast servers can be used to replicate and transmit multicast packets, or the IP multicast model of source-based trees can be used. Different delivery mechanisms may be appropriate for different multicast modes and appropriate multicast data delivery mechanisms are currently being investigated.

After receiving a mesh setup request, Beagle triggers the sending of flow setup messages that reserve resources for flows in the mesh along the path of the flow. All flows within a mesh are set up in parallel, and asynchronous notifications are delivered to the application regarding success/failure of the flow setup process. Flow setup can be initiated either by the sender or by the receiver. Proxy signaling allows flow setup to be initiated by a third party. Beagle protocol mechanisms use a hard-state approach to establish state in routers. The third-party flow setup is the basic mechanism used to setup the application mesh. An efficient setup of the mesh includes the establishment of a *mesh core* which is defined by virtual links between designated routers, plus individual flow setups initiated by senders or receivers, that rendezvous with the core in designated routers.

Figure 4(b) illustrates the message flow for flow-setup initiated by a third-party. We focus on unicast third-party setup for ease of exposition, but the same mechanisms are used for multicast flow setup as well. There are three messages (SETUP_REQ, SETUP_RES and SETUP_CONF) exchanged between neighbor routers as part of the three-way handshake in setting up a flow. Figure 4(a) shows the various objects carried in a SETUP_REQ message. The Application Id object identifies the virtual mesh and the Flow Descriptor, Sender Tspec and Flow QoS objects identify the flow and its QoS requirements. The Route Constraint object carries explicit routing information and the App Flow

Hierarchy and Temporal Sharing objects carry resource sharing information.

Suppose a third-party C wants to establish a point-to-point flow from A to B. C creates a SETUP_REQ message with the appropriate parameters, encapsulates the message in a PROXY_REQ message and transmits message directly to A. A strips off the PRXOY_REQ container and forwards the message towards B. Each router along the path from A to B intercepts the message and allocates resources for the flow as specified in the SETUP_REQ message. If the flow has a specified explicit route, the forwarding path is selected based on the Route Constraint object. When the SETUP_REQ message reaches the destination B, B responds with a SETUP_RES message that proceeds hop-by-hop towards A. At each hop along the path, a node receiving the SETUP_RES message responds by sending a SETUP_CONF message upstream completing the three-way handshake. When A receives SETUP_RES message, it encapsulates it in a PROXY_RES message and sends it to C to complete the third-party initiated flow setup. Flow teardown is accomplished by a two-way handshake using TEAR_REQ and TEAR_RES messages. During the third-party setup, error conditions are also reported to the initiator C. Figure 4(b) shows an example of an end-point `m3` establishing a point-to-point flow between end-points `m1` and `m2`.

4.2 Delegates

The term *delegate* refers to code segments that are defined by applications or service providers and execute in the interior of the network. Delegates are broadly classified as “data” delegates and “control” delegates. Data delegates primarily reside in compute servers and provide data manipulation services such as transcoding, mixing, etc. Control delegates on the other hand, reside primarily on routers or switch nodes and perform resource management tasks such as intelligent adaptation, application-specific re-routing, etc. The example shown in Figure 3(a) shows the use of a data delegate that performs MPEG to Motion JPEG transcoding. Delegates are a focussed application of active networks [11]: network users can add functionality to the network.

In contrast to current signaling protocols like RSVP and PNNI, Beagle allows applications to allocate computation and storage resources required for delegates. Unlike flow setup which involves exchanging of messages between adjacent routers along the path, delegate setup only involves exchange of messages between the originating node and the delegate execution node. The execution node can either be an end-system or a router. Delegates are characterized by their QoS requirements, runtime environment requirements and the resource handles they need to perform resource management tasks. There are currently two defined QoS services for delegates: 1) Guaranteed service — where the delegate is guaranteed to get a fraction (expressed as percentage) of a reference CPU (which is then mapped to the particular CPU on which the delegate executes) and 2) Static Priority service — where the delegate can specify a weight that represents its scheduling priority. In addition to CPU QoS service classes, memory and storage requirements are expressed as a range defined by minimum and maximum values.

Currently delegate runtime environments are characterized by the combination of a *code type* and *runtime type*. Code type identifies the language in which a delegate is coded (e.g., Java, Perl, VisualBasic script, etc.). Runtime type identifies the native library requirements of the delegate (e.g., JDK 1.0.2, WinSock 2.1, etc.). In addition to delegate QoS and runtime requirements, the delegate setup message also contains a list of flow descriptors which identify flows which are manipulated by the delegate at the execution node. At the execution node, when a delegate setup message comes in, Beagle locates the appropriate runtime environment, instantiates the delegate and passes handles to LRM reservation state for all flows that are requested. Using these handles, the delegate can interact directly with the LRM to perform resource management for the flows during runtime.

The representation of the delegate QoS and runtime objects described above represents a first step towards characterizing computational QoS and runtime requirements. Better representations are expected to be developed in the future.

4.3 Resource Sharing

Beagle enables applications to control the distribution of resources amongst the flows comprising the mesh. There are two ways by which the distribution of resources can be controlled. *Hierarchical Sharing* defines a hierarchical distribution of resources amongst flows and *Temporal Sharing* defines an aggregate resource requirement for arbitrary groups of flows within the mesh. Each type of sharing is examined in more detail in the following sections. The two sharing objects together with the concept of the mesh core described in Section 4.1, allow applications to use their high-level knowledge to force co-operative and competitive sharing within the mesh. This results in higher resource efficiency and improved quality for the application.

4.3.1 Hierarchical Sharing

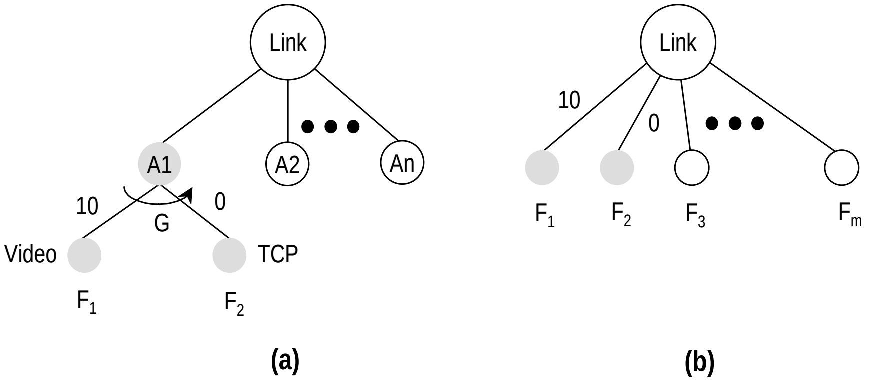


Figure 5: Hierarchical sharing example

Figure 5 shows an example that demonstrates the benefits of hierarchical sharing within an application. Consider an interactive video conference between two parties with a point-to-point MPEG video flow and a point-to-point TCP flow exchanging data. The video flow has a reserved bandwidth of 10 Mbps while the TCP flow is a best-effort flow as shown in Figure 5. Using hierarchical sharing as shown in Figure 5(a), both TCP and video flows can be grouped under the same application. This has the advantage that the bandwidth not used by the video flow is available to the TCP flow. Without hierarchical sharing the unused bandwidth of the video flow is shared amongst all best-effort flows that exist at the link as shown in Figure 5(b). This example clearly demonstrates that hierarchical sharing can be used to achieve co-operative sharing between application flows where best-effort flows can opportunistically use bandwidth leftover by bursty flows.

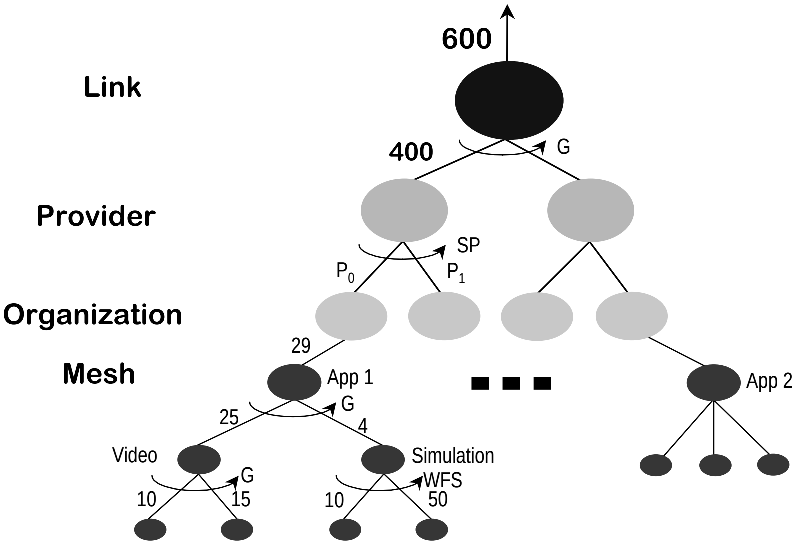


Figure 6: Example of a link resource tree.

As explained in Section 3, Darwin uses hierarchical management of resources at various levels. At a particular link, this takes the form of a resource tree which describes the hierarchical allocation of link resources for all flows at that link. As shown in Figure 6, lower portions of the resource tree consist of sub-trees that express hierarchical resource allocation for flows within an application mesh, at that particular link. However resource trees are very specific to the particular type of hierarchical scheduler at that link (see for example [12], [13]). In order to hide low level details of heterogenous schedulers from applications and resource brokers, a high-level abstraction is needed that expresses the hierarchical sharing within the mesh in a scheduler-independent fashion.

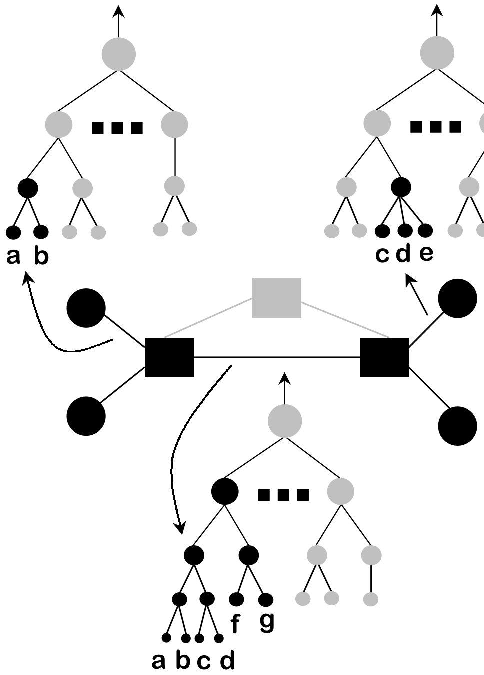


Figure 7: Example application mesh showing various link resource trees. Nodes and links that are part of the mesh are shaded black.

As shown in Figure 7, the hierarchical sharing within the mesh is expressed by the collection of individual link resource trees from the physical links comprising the mesh. The individual link resource trees are different as different application flows share different links, which results in different sharing behavior. On the other hand, the description of an application mesh generated by an application or resource broker contains virtual links. Therefore a high-level abstraction is needed to capture the hierarchical sharing across all links comprising a virtual link.

In order to meet the twin goals described above, Beagle uses the concept of a *hierarchical grouping tree* which expresses the hierarchical sharing across a virtual link. For simpler meshes, it is sometimes possible to encode the entire hierarchical structure within a single grouping tree as shown in Figure 3(a). In order to meet the first goal, the hierarchical grouping tree defines *hierarchical QoS services* by associating generic QoS service types with interior nodes of the tree. The leaf nodes of the trees represent the flows whose QoS requirements are expressed by their individual flowspecs. Service-specific rules describe how child node flowspecs are aggregated into the parent node flowspec. Using QoS service types of the interior nodes, QoS flowspecs of the leaf-nodes (flows) and service-specific aggregation rules, individual link resource sub-trees can be derived depending on the type of hierarchical scheduler at that link. This involves converting flowspecs at each node into appropriate low-level scheduler-specific parameters for that node in the scheduler hierarchy. For example, in the case of Hierarchical Weighted Fair Share schedulers [12] the appropriate parameter is a scheduling weight associated with each node, and in the case of Hierarchical Fair Service Curve [13] the appropriate parameters are the three parameters used to represent a service curve at each node. In order to meet the second goal, the hierarchical grouping tree encodes the hierarchical sharing structure of all flows sharing a virtual link. Based on application flows that share a particular physical link in the network, the hierarchical grouping tree can be pruned to eliminate flows that don't exist at that link.

4.3.2 Temporal Sharing

The mesh is a convenient vehicle for indicating application-specific opportunities for resource optimization. Specifically, 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 no more than 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. Beagle provides a flexible way for applications or resource brokers to specify this application-domain information through a temporal sharing object that lists a set 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 generalizes RSVP's notion of resource reservation styles. In RSVP flowspecs can only be aggregated by either a sum or a least upper bound (LUB) operation on individual flowspecs. Another limitation of RSVP is that sharing is limited to flows within a multicast session. In Beagle, the temporal sharing object can be used to group arbitrary flows within an application mesh and an arbitrary flowspec can be associated with the group of flows. As shown in Figure 3(a), the distributed interactive simulation application associates an aggregate Controlled Load service flowspec with the two simulation flows.

A common application that benefits from temporal sharing is a controlled multimedia conference where only a small set of participants are active at one time. Suppose for example, that only two video sources characterized by token buckets (p_1, r_1, b_1) and (p_2, r_2, b_2) can be active at a time. If it is known that bursts from the two sources do not appear back-to-back, then temporal sharing can be used to make an aggregate reservation for the traffic characterized by $(\max(p_1, p_2), r_1 + r_2, \max(b_1, b_2))$. Without this knowledge the aggregate reservation would be the sum of the two token buckets $(p_1 + p_2, r_1 + r_2, b_1 + b_2)$. This results in unnecessary over allocation of resources.

4.3.3 Combining Hierarchical and Temporal Sharing

The hierarchical grouping 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 can be used independently of each other. If both types of sharing are specified, the resource sub-tree is derived by applying the temporal sharing specification at every level of the tree. In cases where the temporal sharing object lists flow groups that do not fall under the same parent node in the resource sub-tree, the temporal sharing behavior is ignored. The example shown in Figure 3(a) shows the combined use of both sharing objects. The resulting link resource sub-trees that results at links L_1 and L_2, assuming the use of Hierarchical Weighted Fair Share schedulers [12], are shown in Figure 3(b). As seen in Figure 3(b), the aggregation for Guaranteed service video flows results in the summation of individual reservations, while using temporal sharing an aggregate Controlled Load flowspec is associated with the simulation flows. On link L_1, the simulation flows F_3 and F_4 share the aggregate rate in a Weighted Fair Share fashion (with weights 10 and 50), while at link L_2 the grouping tree is pruned to eliminate flow F_4 thereby allocating the complete Controlled Load bandwidth to flow F_3. These resource sub-trees then attach to the service provider or organization from which resources are drawn at that link.

5 Beagle Prototype

A prototype of the Beagle resource allocation protocol has been implemented on the Darwin and CAIRN [14] testbeds. The Darwin testbed topology is shown in Figure 8. This section describes the implementation of the Beagle prototype system and presents preliminary performance evaluation results. The Beagle prototype supports sender-initiated, receiver-initiated and third-party point-to-point flow setup, delegate setup and has limited support for hierarchical resource sharing. The prototype does not currently support multicast flow setup and temporal sharing.

5.1 Beagle Implementation

Figure 9 shows the implementation of the Beagle prototype system on a router and host, both of which are UNIX workstations. This implementation is based on the RSVP implementation distributed by ISI (available from `ftp://ftp.isi.edu/rsvp/release`) and runs on several UNIX platforms such as Digital Unix, FreeBSD, NetBSD and Linux. Applications and/or Xena can invoke Beagle by making appropriate API function calls. The API (shown by the shaded portions in Figure 9) is a library that is compiled in as part of the application. It communicates with the Beagle daemon through UNIX domain sockets. Important API calls include creation of a mesh, flow or delegate and registering a delegate runtime and listening for incoming delegates. The API also has calls to attach to an existing mesh and to dynamically modify flow characteristics during runtime. The Beagle daemon communicates with other Beagle daemons using raw IP.

The current implementation uses a hard-state approach based on *three-way handshakes* between adjacent nodes to setup flows. Currently, timers associated with standard three-way handshakes are used to recover from failures due to

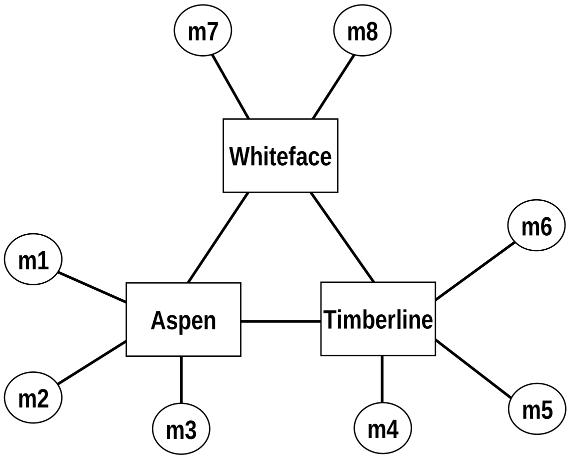


Figure 8: Darwin testbed topology. The testbed currently consists of three Pentium II 200 MHz PCs configured as routers running NetBSD 1.2D and several end-systems running several variants of Unix. All links are point-to-point Ethernets configurable as 10 or 100 Mbps.

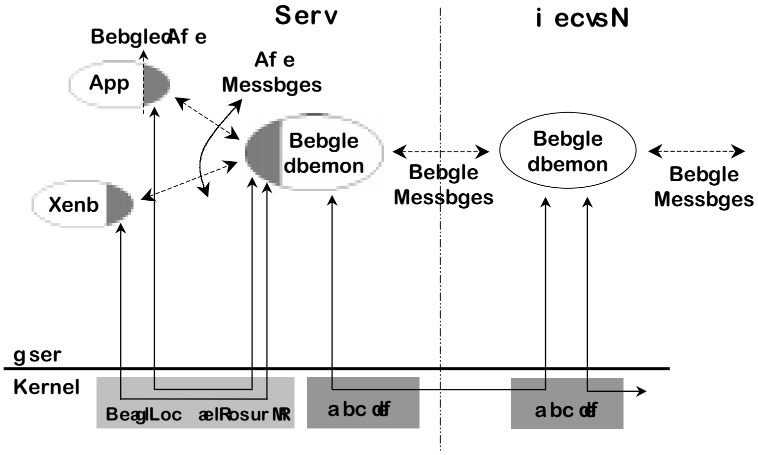


Figure 9: Beagle prototype and Beagle API implementation.

packet drops. Future implementations will use a simple reliable transport protocol on top of raw IP to ensure delivery of Beagle messages. This reliable transport protocol can either be TCP or some other lightweight mechanisms such as those used with LDP [15] or ATM PNNI [5]. While both soft-state and hard-state approaches can be used to establish state, they have different trade-offs in the areas of implementation complexity and error recovery. In the context of the Internet, the fall-back position in the event of failures, is to rely on best-effort connectivity. Mechanisms for full recovery of application mesh state will be explored in the future. It is expected that a combination of soft-state and hard-state approaches might work well for recovering different parts of mesh state.

5.2 Beagle Mesh Representation

Figure 10 shows the distributed application mesh state maintained by Beagle in a router. The `appMesh_t` data structure shows the various components of an application mesh. The mesh is identified globally by the `appId` field which is a concatenation of the IP address of the owner node and a serial number allocated by the owner. The mesh also has a linked list of flows and delegates which are part of the mesh. The `afh` and `ts` represent the grouping tree and temporal sharing objects respectively. The grouping tree object `groupTree_t` is basically a variable length

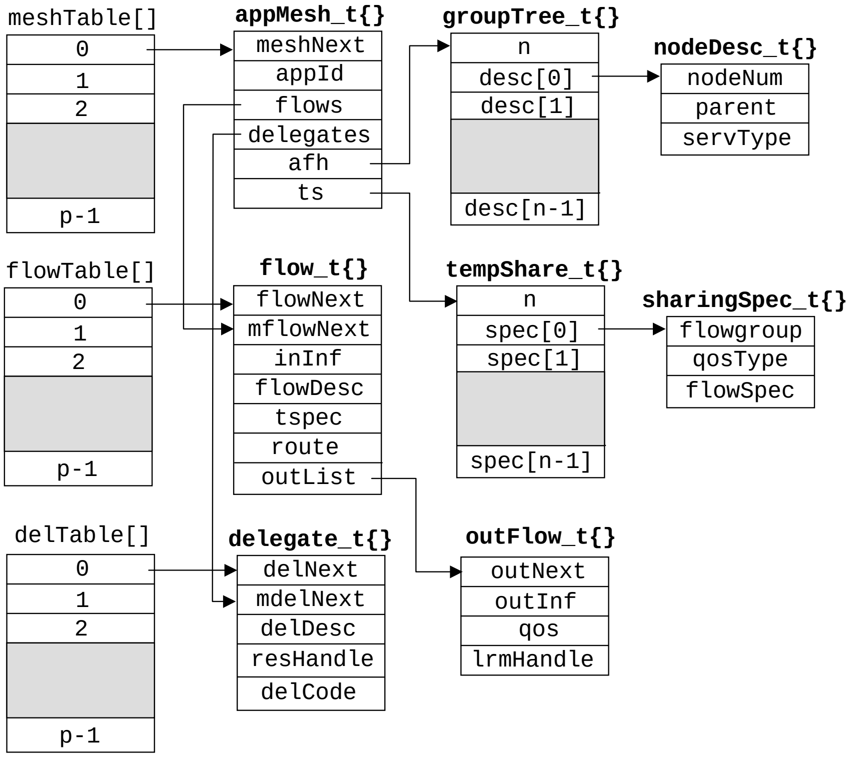


Figure 10: Representation of an application mesh in the Beagle prototype implementation.

list of node descriptors describing the nodes in the grouping tree. Each node is identified by the triplet `(nodeNum, parent, servType)`. `servType` is one of the four basic QoS service types supported in Beagle as described in Section 4.1. The temporal sharing object `tempShare_t` is also a variable length list of flow combinations and their associated aggregate flowspecs. Each flow is characterized by the `flowDesc` which is the globally unique flow descriptor as described in Section 4.1. The flow object `flow_t` represents a one-to-many multicast flow. Each output branch of the flow at a router has an associated QoS flowspec and a local resource manager handle to the reservation. A delegate is characterized by the triplet `(delDesc, resHandle, delCode)`. `delDesc` identifies the QoS requirements of the delegate, `delCode` contains the executable code and the runtime requirements and `resHandle` contains the handles to resources requested by a delegate as explained in Section 4.2.

5.3 Performance Evaluation

This section reports results of experiments conducted to measure the performance of Beagle prototype implementation. The implementation runs on the Darwin testbed as shown in Figure 8. The experiment involved setting up the application mesh discussed in the example. End-to-end setup latencies for flows and delegates, and per-router flow setup processing times were measured. The results are tabulated in Tables 1 and 2. All measurements reported are averages from 100 runs.

Flow	**Delegate**	**Mesh**
7589	3811	18989

Table 1: End-to-end average setup latency in microseconds. Flow setup latency is through two routers.

In the current prototype implementation, multiple flow setup requests originating from the same node cannot be issued in parallel. Also, delegate setup can only be initiated after the flows have been setup. Therefore, application mesh setup time for the example mesh is two times the flow setup time plus the delegate setup time. This can be improved by simultaneously issuing all the setup requests as discussed in Section 4.1.

Route Lookup	LRM Interaction	Message Parsing	Message Processing	Total
145	1578	339	1997	2336

Table 2: Break-up of flow setup processing time in microseconds at a router. Message processing time includes both route lookup and LRM interaction.

As shown in Table 2, about 68% of the per-router flow processing time is spent in interacting with the local resource manager. This involves admission control, and setting up flow state in the packet classifier and scheduler. An estimated As seen from Table 2, the current Beagle prototype supports about 425 flow setups per second. This is comparable to connection setup times reported for various ATM switches.

6 Related Work

The development of QoS-capable networks like ATM and Integrated Services Internet has caused a boom in the area of signaling research. As a result, several resource allocation protocols capable of supporting applications that request per-flow QoS have been developed. This section briefly describes a few of these protocols and contrasts them with Beagle.

In the context of Integrated Services Internet, RSVP [4] and SCMP [3] are the two resource allocation protocols that have been developed. The more popular of these protocols, RSVP, has been explicitly designed to support multimedia multicast applications and uses a receiver-oriented resource allocation model. RSVP allocates resources on a per-flow basis, where a flow is either a unicast session or an entire multicast session or a sender-specific multicast flow within the multicast session. RSVP uses soft state with periodic refreshes to establish reservation state. SCMP on the other hand was initially designed with the sender-oriented resource allocation model for unicast flows, but was later extended to include both sender and receiver oriented reservations and also multicast. SCMP uses hard state to establish reservations. In the context of ATM networks, the Private Network-Network Interface [5] protocol was developed to setup a virtual circuit with a specified QoS between ATM switches on demand. PNNI also allocates resources on a per-flow basis where a flow is either a virtual circuit (VC) or a virtual path (VP). Like SCMP, PNNI uses hard state. While the protocols mentioned above were developed primarily within standards organizations like IETF and ATM Forum, there have been many experimental protocols developed by research institutions or universities. Some of these are the Tenet protocols [16] [17], CMAP/CNMP [18] and DCPA [19]. All these protocols are similar to PNNI in the sense that they allocate resources on a per-flow basis and use hard-state.

Beagle differs from the protocols mentioned above in several respects. Beagle uses the application mesh as the basic unit for resource allocation. This provides great flexibility for structured multi-party, multi-flow applications to optimize resource allocation within the confines of the application mesh. Beagle also uses a multicast model that is more controlled with respect to the IP multicast model where receivers can arbitrarily join and leave multicast groups. The current implementation of Beagle uses hard state for establishing reservation state, but can easily be adapted to use soft-state. Beagle supports both receiver-oriented and sender-oriented resource allocation.

While most research has focused on resource allocation for communication resources, emerging applications will need to allocate computational resources as well. A few protocols have been developed that provide varying degrees of support for delegates. X-Bind [20] defines a object-oriented programming model where applications can allocate computational resources. However it does not provide support for downloading delegates into the network and providing them with appropriate handles for resource management during runtime. On the other hand the Connection Closures work [21] which provides support for downloading delegates, does not address the issue of allocating computational resources for their operation. Beagle provides support for both of the issues raised above.

The recent development of the differential services architectures [22] [23] has brought to the forefront the need for aggregate resource allocation in the core of the Internet for better scalability. It has also identified the need for a resource allocation protocol capable of supporting aggregation. Several protocols support aggregation to varying degrees. Recently, extensions have been proposed to RSVP [24] [25] [26] that advocate the use of CIDR style filters to support aggregation and propose mechanisms to allow core routers to process only aggregated resource allocation requests. Beagle incorporates similar mechanisms more naturally based on its application mesh model.

The issue of resource optimization via sharing has largely been overlooked. RSVP provides some support for resource sharing within a multicast session by providing different reservation styles. These styles control how resources can be shared among different sender-specific flows in a multicast session. Beagle generalizes this concept further by providing mechanisms through which resource dependencies among flows within an application can be expressed and exploited.

7 Conclusions

This paper presented the design and implementation of the Beagle resource allocation protocol, which allows an emerging class of multi-party, multimedia, multi-flow applications to optimize resource allocation using high-level application domain knowledge. This paper demonstrates that using Beagle to propagate high-level application structure to appropriate entities in the resource management chain, enables applications to achieve higher quality service, while at the same time improving resource efficiency for the network. This is in contrast to RSVP which addresses an application-domain which primarily consists of large-scale unstructured multicast applications. Beagle provides mechanisms through which such applications can allocate computation and communication resources, tailor resource management during runtime by using delegates, aggregate resource allocation over varying granularities and optimize resource allocation by expressing a broad range of resource sharing behavior. Beagle addresses the need for a resource allocation protocol that takes an integrated view of an application in the context of emerging advanced services Internet.

Although initial results from a prototype implementation are promising, much work remains to be done. Scalability needs to be addressed further by defining mechanisms that allow aggregation within the core of the Internet across different application meshes. Parameters used to express computational QoS and characterize delegate runtime requirements require further study. A naming scheme is required to address delegates and identify their resource requirements. Mechanisms need to be defined to co-ordinate inter-delegate communication during runtime. Delegates raise security concerns and explicit authentication, authorization and encryption methods are needed. Better and more intuitive representation of sharing behavior needs to be explored.

References

[1] Stephen E. Deering and David R. Cheriton. Multicast Routing in Datagram Internetworks and Extended LANs. *ACM Transactions on Computer Systems*, 8(2):85–110, May 1990.

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

[3] L. Delgrossi and L. Berger. Internet stream protocol version 2 protocol specification - version st2+, August 1995. Internet RFC 1819.

[4] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource Reservation Protocol (RSVP) – Version 1 Functional Specification, September 1997. IETF RFC 2205.

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

[6] Prashant Chandra, Allan Fisher, Corey Kosak, T.S. Eugene Ng, Peter Steenkiste, Eduardo Takahashi, and Hui Zhang. Darwin: Resource Management for Value-Added Customizable Network Services. In *Sixth International Conference on Network Protocols*, Austin, October 1998. IEEE.

[7] Prashant Chandra, Allan Fisher, Corey Kosak, and Peter Steenkiste. Network Support for Application-Oriented Quality of Service. In *Proceedings Sixth IEEE/IFIP International Workshop on Quality of Service*, pages 187–195, Napa, May 1998. IEEE.

[8] V. Fuller, T. Li, J. Yu, and K. Varadhan. Classless interdomain routing (cidr): an address assignment and aggregation strategy, September 1993. Internet RFC 1519.

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

[10] J. Wroclawski. Specification of the Controlled-Load Network Element Service, September 1997. IETF RFC 2211.

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

[12] J.C.R. 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.

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

[14] Collaborative advanced interagency research network (cairn). http://www.cairn.net.

[15] E. Rosen, A. Viswanathan, and R. Callon. A proposed architecture for mpls, August 1997. Internet draft, draft-ietf-mpls-arch-00.txt, work in progress.

[16] Anindo Banerjea, Domenico Ferrari, Bruce Mah, Mark Moran, Dinesh C. Verma, and Hui Zhang. The tenet real-time protocol suite: Design, implementation, and experiences. *IEEE/ACM Transactions on Networking*, 4(1):1–10, February 1996.

[17] B. Bettai, D. Ferrari, A. Gupta, W. Heffner, W. Howe, M. Moran, Q. Nguyen, and R. Yavatkar. Connection establishment for multi-party real-time communication. In *Proceedings of the 5th International Workshop on Network and Operating System Support For Digital Audio and Video*, pages 255–266, Durham, New Hampshire, April 1995.

[18] K. Cox and J. DeHart. Connection management access protocol (cmap) specification. Technical Report WU-CS-94-21, Department of Computer Science, Washington University, November 1994.

[19] M. Veeraraghavan and T. F. La Porta. Object-oriented analysis of signaling and control in broadband networks. *International Journal of Communication Systems*, 7(2):131–147, April 1994.

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

[21] S. Rooney. Connection closures: Adding application-defined behavior to network connections. *Computer Communication Review*, 27(2), April 1997.

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

[23] 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.

[24] Jim Boyle. RSVP Extensions for CIDR Aggregated Data Flow, June 1997. Internet draft, draft-ietf-rsvp-cidr-ext-01.txt, work in progress.

[25] S. Berson and S. Vincent. Aggregation of internet integrated services state, November 1997. Internet draft, draft-berson-classy-approach-01.ps, work in progress.

[26] R. Guerin, S. Blake, and S. Herzog. Aggregating RSVP-based QoS Requests, November 1997. Internet draft, draft-guerin-aggreg-rsvp-00.txt, work in progress.