A Signaling Protocol for Structured Resource Allocation

Prashant Chandra†, Allan Fisher‡ and Peter Steenkiste‡
† Electrical and Computer Engineering, ‡ School of Computer Science
Carnegie Mellon University
Email: {prashant,alf,prs}@cs.cmu.edu

Abstract— **There is an emerging class of multi-party, multimedia, multi-flow applications 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 places new requirements on signaling protocols. This paper outlines these new requirements, discusses ways in which they can be supported and presents the design and implementation of an experimental signaling protocol that supports these requirements.**

This paper makes the case that for these structured applications, there is an advantage to allocating the resources in an integrated fashion, i.e computation, storage and communication resources for all the flows are allocated at the same time in a coordinated fashion. The concept of a *virtual mesh* is introduced as a key abstraction that encapsulates the set of resources that are allocated and managed in an integrated fashion to meet the needs of applications. The paper presents two mesh setup algorithms and a performance evaluation comparing them. *Temporal resource sharing* within the virtual mesh is discussed in detail and signaling support for temporal sharing at setup and runtime is examined. It is important to characterize temporal sharing since it can significantly reduce the resource requirements for applications.

We have implemented the *Beagle* signaling protocol that supports this integrated resource management model. Beagle representations and mechanisms for mesh setup and temporal sharing are described and a prototype implementation is presented.

I. Introduction

Advances in networking research and technology have made high-speed networks that provide various qualities of service a reality. At the same time, the increasing popularity of the world wide web has increased interest in the development of sophisticated, multimedia applications that can make use of the features these advanced networks offer. We expect to see increasingly sophisticated multimedia, multipoint applications that use multiple flows. Examples include video conferencing, interactive games and distributed simulation.

On the other hand, current resource management architectures and protocols continue to use the per-flow resource allocation model. Moreover, existing signaling protocols allocate only communication resources, while emerging applications use a much broader set of resources, including computation and storage. This paper makes the case that for these structured applications, it is advantageous to allocate the resources in an integrated fashion, i.e computation, storage and communication resources for all the flows are allocated in coordination. This approach has the advantage that it allows the use of high-level global knowledge about the application and the network to achieve improved quality and resource efficiency, and it also allows a degree of customization of the allocation and management of resources by applications.

We introduce the concept of a *virtual mesh* as the key abstraction that encapsulates the set of resources that are allocated and managed in an integrated fashion to meet the needs of applications. Similar to per-flow based resource allocation, mesh-based allocation has two components: a resource broker [1] that identifies the resources needed to satisfy an application request (similar to routing), and a signaling protocol that actually allocates the resources (similar to per-flow signaling protocols such as RSVP [2], PNNI [3]). This paper focuses on the second component. We present a set of features that are necessary in mesh-based signaling protocols, describe why these features are important and how they can be supported, and discuss how they can improve performance relative to per-flow based allocation.

We also describe the design and prototype implementation of a resource allocation protocol called Beagle, intended for structured applications in integrated services networks. Beagle has been developed in the context of the CMU Darwin project [4]. Beagle allows applications to customize resource allocation during setup, and resource management during runtime. In this paper we focus on two aspects of Beagle: 1) virtual mesh setup and 2) resource sharing among flows within a virtual mesh. We argue that both aspects of the signaling protocol benefit substantially from the use of structured resource management.

The rest of the paper is organized as follows. Section II outlines the motivation for this work. Section III describes our overall resource management architecture for structured applications. Section IV describes the specification of the virtual mesh. Resource sharing is discussed in Section V. Mesh setup algorithms are evaluated in Section VI. Section VII describes the design and implementation of Beagle. Section VIII contrasts our approach with related work and Section IX presents the conclusions.

II. Motivation

Current resource allocation protocols for integrated services networks use the "flow" as the basic unit of resource allocation. Flows can be end-to-end application flows at the lowest granularity, or can be aggregated flows such as virtual links leased by an organization. These protocols work well for simple applications with a single flow or a small number of flows. However there is an emerging class of complex multimedia, multi-party applications that use many flows of various data rates and QoS requirements. These applications have a high-level struc-

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

ture 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 cannot address the needs of this class of applications.

Let us use a simple motivating example of a distributed interactive simulation to identify some of the resource management requirements for such applications. This application has a many-to-many communication pattern with bandwidth requirements on both video and voice delivery among the participants, and on data delivery both among simulation end-points and between simulation endpoints and participants. Information on the structure of this application opens the door for several resource allocation optimizations:

- *Coordinated resource allocation* — 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 placement of the servers will depend on the availability of communication resources in the network, and the selection of communication resources will depend on server placement. In other words, these diverse resources should be allocated in a coordinated fashion.
- *Flow manipulation* — the application may need to use generic transcoding services to perform video type matching among participants. It may also use video mixing, down-sampling and other kinds of flow manipulation services. This requires the allocation of computation and storage resources inside the network in addition to and in coordination with communication resources. The distinction from the previous example is that this can be done transparently to the application.
- *Resource sharing* — the application exhibits a high-level structure that defines how many participants can be active in the video conference at one time, opening the door for resource sharing. 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 achieve more efficient resource sharing within the application. Bandwidth sharing inside the application can be increased by routing groups of multimedia flows together to promote better sharing and synchronization.
- *Topology customization* — while simple point-to-point applications in general do not care about the route taken by their data, more sophisticated multi-point application may have application-specific constraints on how flows should be routed. One example is the routing of flows together to promote resource sharing, as mentioned above. Other examples of topology customization are using disjoint routes to improve reliability, and using multi-path routes for best-effort flows to improve throughput.

Another set of resource management requirements are driven by the network. Besides the more traditional requirements, e.g. dealing with multiple service classes, we identify the following important requirements:

- *Scalability* — the main scalability consideration is not so much the number of end-points, since most applications will involve a relatively modest number of nodes, i.e. 10s to 100s but not 1000s, but the number of applications that must be supported. This means that the overhead to establish a session on core network elements should be minimized. One example of such an optimization is to move overhead as much as possible to end-points and servers. Another important example is aggregation: applications such as large distributed interactive simulations exchange data between several sites, with several participating hosts per site. This results in a large number of flows traveling between the same pairs of sites. Aggregating these flows can reduce resource allocation and runtime management overhead.
- *Mixed resource allocation* — meeting end-to-end requirements may not require the same type of resource allocation (i.e. service type) along the entire path. For example, reservations may be needed only on certain congested segments. The same service classes may in fact not even be available along the entire path. The signaling protocol must be able to deal with such mixed resource allocation requests.

These application and network requirements can best be realized with a resource allocation architecture that takes an integrated view of the application, and can propagate and exploit its structure at appropriate times during resource allocation process. In the next section we describe an architecture that supports structured resource allocation, and the remainder of the paper focuses on the signaling protocol.

III. Structured Network Resource Management

A. Architecture

Our architecture has two components. The first component, the resource broker [1], identifies the resources that best satisfy a given application request. It can be viewed as an extension of traditional routing protocols. The second component is the signaling protocol. It is responsible for allocating the resources by contacting the local resource managers for each of the required resources.

The above architecture is driven in part by the goal of delivering customizable services in a heterogeneous network environment that includes multiple administrative domains. Resource brokers interact directly with applications, and can incorporate knowledge about the application domain. A resource broker specializing in video conferencing could, for example, receive a high level request such as “high-quality video conference for 5 participants.” Given the location of the participants, it could identify the resources that best satisfy the request, performing optimizations such as inserting transcoders transparently to the application. Resources brokers could execute as part of the applications (e.g. a middle-ware library), as part of a service provider, or as a separate service in the network.

In contrast, local resource managers are tied to a particular network resource and administrative domain, and different networks are likely to have different local resource managers that potentially provide different communication services. It is the task of the signaling protocol to bridge the gap between the application-domain oriented resource brokers and the network-specific local resource managers. The resource broker gives the signaling protocol a specification of the resources requested by the application. The specification is not for a single flow, but for a set of flows. We will refer to this set of resources as a *virtual mesh*. The signaling protocol contacts the appropriate local resource managers, maps the high level specification provided by the broker onto the interface provided by the local resource manager, and requests the resources. This process is the focus of the remainder of this paper. The resource management architecture outlined here has been realized in the context of the Darwin project at CMU and is described in greater detail in [4].

The above architecture meets the requirements outlined in the previous section. The resource broker can perform high-level optimizations such as topology customization, since it has access to information on both the application's needs and the network's status, and it can decide on which network segments reservations are required. The resource broker and signaling protocol jointly can set up optimizations such as flow transformations, resource sharing, and aggregation. Moreover, brokers can run outside the core network.

B. Signaling Protocol Requirements

The resource management needs of emerging sophisticated, structured applications place new requirements on the signaling protocol. To maximize its effectiveness, the signaling protocol needs to:

- allocate a broader set of resources including computation and storage resources in addition to communication resources.
- support explicit routing to enable topology customization by applications and resource brokers.
- provide a flexible interface to express resource sharing policies and provide mechanisms for the realization of those policies.
- support resource allocations for traffic aggregates of varying granularities.
- support non-uniform QoS model for flows.
- take an integrated view of the entire application as opposed to individual flows.

In this paper, we focus on allocation and management of network communication resources; allocation of computation and storage resources is not discussed further.

IV. Virtual Mesh

The virtual mesh is a key abstraction that encapsulates the set of resources that are allocated and managed in an integrated fashion to meet the needs of an application. The physical resources include the communication resources required for the set of flows and the computation resources required at end-points and computation servers in the network. By specifying a customized virtual mesh, applications or brokers can implement the resource optimizations outlined in Section II. For example, they can influence how flows are routed, how they share resources with other flows within the mesh and what kind of data transformations they undergo as they traverse the network.

In addition to the resource requirements of an application, the virtual mesh specification also includes a small set of *designated routers* that correspond to merge and split points for flows belonging to the virtual mesh, plus a set of *virtual links* that connect the designated routers. Designated routers correspond to specific routers in the network, while virtual links are typically mapped onto multi-hop paths by the signaling protocol. The placement of the designated routers and the definition of the virtual links support the topology customizations discussed earlier.

Figure 1 shows an example of a virtual mesh and associated network topology of the application. As shown in the figure, the virtual link between the two designated routers corresponds to the network of three routers shown in the shaded portion of the figure. The part of the actual network within the designated routers is called the *virtual mesh core*. Flows are aggregated to varying degrees within the mesh core to improve scalability. We distinguish between two types of aggregation: 1) *state aggregation* determines whether the data plane state (packet classifier and scheduler state) is aggregated for flows which share a link in the core, and 2) *route aggregation* determines whether flows that share virtual links also share physical links within the core between designated routers. Figure 1 demonstrates the two types of aggregation within the mesh core. The two types of aggregation have an effect on the performance of the mesh setup, as shown in Section VI.

Communication resources within the virtual mesh are represented by flows and their QoS requirements. Flows can be unicast or multicast, and are defined by a globally unique *flow descriptor object*, which is a combination of the source and destination IP addresses and port numbers. Beagle flows can represent traffic at various granularities. End-to-end flows require a full specification of the all the flow descriptor fields. Aggregate flows are specified either by using a list of individual flow descriptors, or by using CIDR [5] 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

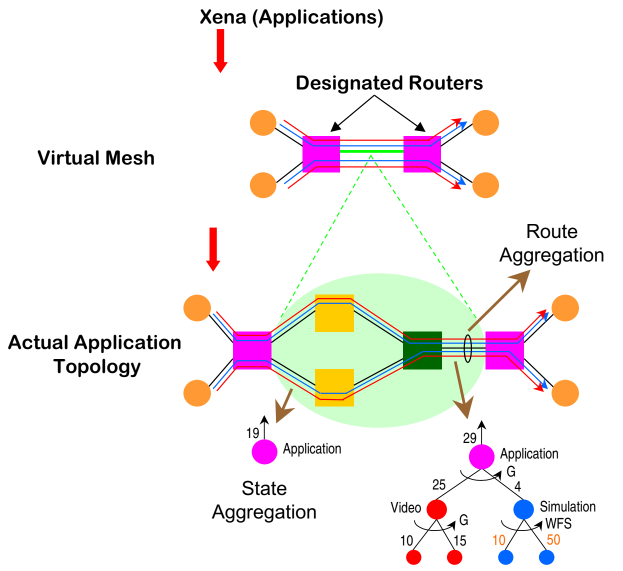

Fig. 1. Example virtual mesh showing route and state aggregation in the mesh core.

aggregation and de-aggregation points respectively. The (beginIP, endIP) pair can also used to meet the mixed-resource allocation requirement outlined in Section II, by constructing a flow with several flow segments with differing QoS requirements. 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.

A virtual mesh is identified globally by an *application id object* which is implemented as a concatenation of the IP address of the node that creates the mesh and a serial number allocated by that node. This unique id can be used to identify the mesh at runtime. This is used, for example, by endpoints that want to join or leave the mesh at runtime. This allows structured mesh creation at startup time to be combined with the flexibility of uncoordinated joins during runtime. The latter is the primary mechanism used by RSVP. By supporting both structured and uncoordinated resource allocation, we accommodate a large set of applications in a flexible way.

In the next two sections we focus on two resource management optimizations: how can flows share resources, and how can we optimize mesh creation.

V. Temporal Sharing

The bandwidth used by flows will often change over time, and many emerging applications have multiple flows whose peaks in activity are interleaved. These flows can potentially share the same set of resources over time. We call this type of behavior *temporal sharing*. It is important to characterize temporal sharing since it can significantly reduce the resource requirements for applications. Temporal sharing can arise either as an inherent property of the application, or can be artificially imposed in order to save resources. An audio conferencing application with some form of "floor control" is an example where temporal sharing is inherent in the application itself, because only one participant can talk at a time. On the other hand, a distributed interactive simulation is an example where temporal sharing can be deliberately imposed to save resources. In this example, a receiver simultaneously receives the output of a simulation and a "talking head" video stream. When the simulation is bursting results, the receiver may be willing to reduce the quality of the video, even though the two sources are independent. In this section, we examine temporal sharing in detail — we describe the different types of temporal sharing and show how they can be specified, discuss what mechanisms can be used to enforce the sharing behavior during runtime, and present the design of temporal sharing support in Beagle.

A. Sender-based Versus Receiver-based Sharing

We distinguish between two types of temporal sharing: sender-based and receiver-based. *Sender-based* temporal sharing occurs as a result of the peaks of activity of different senders being interleaved in time. This can arise naturally as inherent application behavior or can be artificially imposed to save resources as mentioned earlier. *Receiver-based* temporal sharing refers to the ability of a receiver to switch over time among a set of senders from which it can receive data. This is useful in saving resources, because the receiver need not reserve resources for all senders in which it is interested. Instead, it reserves an aggregate flow with enough resources to handle the highest bandwidth sender, and switches among the senders at runtime. Note that source and receiver-based sharing cannot always be cleanly separated since sources and receivers can coordinate their activities.

Figure 1 shows how these pieces fit together. It shows a virtual mesh and the mesh core where multiple application flows come together and share links. It also shows source and receiver segments that either feed into the core or forward data from the core to a specific receiver. Source-based sharing primarily affects the core and the feeder segments, while receiver based sharing primarily affects the core and the receiver segments.

Note that there is an alternative to source or receiver based sharing: sources or receivers could tear down an old reservation and set up new reservations when communication needs change. This approach has the advantage that it may have lower resource use: in a number of cases, it allows unused resources in the source segment to be reclaimed. The disadvantage is that it involves more overhead and that it may make performance less predictable: resources that were given up may not be available when they are needed later on during application execution.

Temporal sharing has long been recognized as an important property, and has been supported in various forms in RSVP [2] and Tenet-2 [6]. Neither of these signaling protocols, however, differentiates between the above cases. The importance of the distinction is shown in the following example. Figure 2 shows a network including three sources, S1, S2 and S3, each send-

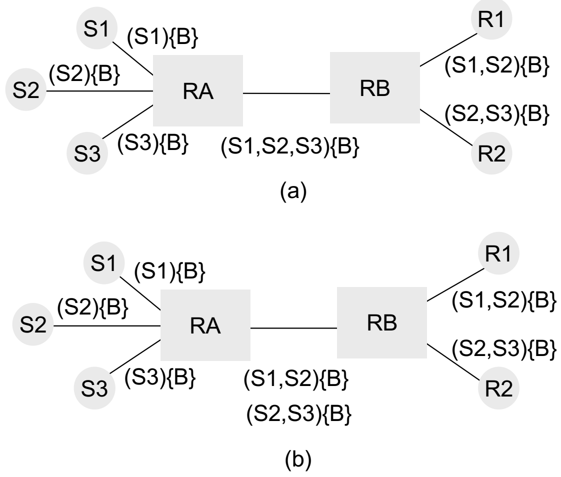

Fig. 2. (a) Sender-based temporal sharing (b) Receiver-based temporal sharing

ing data that requires one unit of bandwidth, B. Two receivers, R1 and R2, receive data from the three sources. Receiver R1 wishes to receive sources S1 and S2, and wants to allocate a total bandwidth of B. We denote this as (S1, S2){B}. Similarly, receiver R2 specifies (S2, S3){B}. In the sender-based case, only one sender is active at a time. This results in an aggregate reservation of (S1, S2, S3){B} on the link between routers RA and RB as shown in Figure 2(a). In the receiver-based case, the reservations cannot be merged because all the senders are simultaneously active. Therefore, two reservations (S1, S2){B} and (S2, S3){B} need to be established on the inter-router link as shown in Figure 2(b).

The RSVP and Tenet-2 sharing models support only the sender-based case. While this is sufficient for certain types of applications, e.g. MBone, other applications such as our earlier example can benefit from both types of temporal sharing. By having separate specifications for sender-based and receiver-based temporal sharing, Beagle supports a more general form of resource sharing.

B. Runtime Enforcement Mechanisms

In addition to setup-time support (described in the previous section), signaling protocols may also have to support runtime mechanisms, such as enforcing temporal sharing and changing the use of resources by flows. Neither RSVP nor Tenet-2 address the runtime aspects of temporal sharing.

There are several ways of enforcing temporal sharing behavior at runtime. In the case of receiver-based temporal sharing, enforcement happens through "resource switching." Resource switching determines which sender is granted use of the reserved resources. Resource switching can be done inside the application on an end-to-end basis (point-to-point flows), or it may require receiver-network interactions to select which data is delivered to the receiver (multicast flows). In the latter case, a new Flow Descriptor may have to be associated with the allocated resources, and the data forwarding changed using a protocol such as IGMP [7].

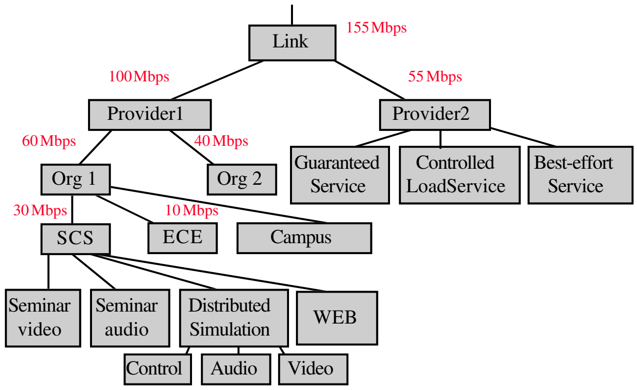

Fig. 3. Example of a link resource tree.

Sender-based temporal sharing presents several options for runtime enforcement. First, sources may coordinate directly among each other, e.g. a source that wants to ramp up first requests some other sources to scale back. This is possible and somewhat similar to natural source based sharing, but it is tedious and error-prone for applications. Second, the network can explicitly manage bandwidth on a per flow basis and application sources can request the network to change the bandwidth distribution. This would for example be appropriate for guaranteed flows. Another option is for sources to coordinate indirectly through the network. For example, suppose a network enforces resource allocation based on both flows and flow aggregates, as in the case of hierarchical scheduling [8]. If one flow in an aggregate ramps up, it will reduce the resources available to the other flows. This will cause congestion and packet drops, which will cause the other flows to back off, for example using protocols such as TCP. This option is appropriate for best effort flows, which are likely to continue to constitute the lion's share of network traffic.

Hierarchical resource management [4] is a promising mechanism to support artificially imposed temporal sharing in an application. Hierarchical resource management is based on the use of hierarchical schedulers that schedule packets based on a resource tree specification. The resource tree (Figure 3) specifies a hierarchical relationship between flows and flow aggregates. A key feature is that reservations can be associated not only with individual flows (leaves of the tree), but also with flow aggregates (interior nodes). The flows sharing resources can be grouped under a single node in the resource tree. That node has a pool of resources associated with it that can be shared by the flows. The flows can include both guaranteed flows and best effort flows that dynamically share the remaining bandwidth in the pool. Support for hierarchical resource management is in fact one of the motivations for the development of Beagle in the Darwin project.

As discussed above, runtime enforcement mechanisms can take several forms, and can depend on both the application and the network. The key is that the signaling protocol should not enforce a single model of runtime resource sharing. Signaling protocols must support both setup-time resource allocation (sender-based and receiver-based sharing) and runtime enforcement to fully exploit temporal sharing.

C. Beagle Temporal Sharing Support

Beagle supports sender-based temporal sharing by using a list of flow combinations and their associated aggregate QoS requirements. In the case of sender-based temporal sharing, flow setup proceeds in the direction of sender to receiver. The list of flow combinations is expressed using the Temporal Sharing object that is carried as part of the SETUP_REQ message. Applications and/or resource brokers can list any number of flow combinations in the temporal sharing object. At each link, Beagle determines the aggregate resource requirements for the group of mesh flows sharing that link, by searching the list of specified flow combinations for the combination that minimizes the resource allocation for that group. The temporal sharing object generalizes RSVP's notion of "resource reservation styles." In RSVP, QoS requirements can be aggregated only by either a sum or a least upper bound (LUB) operation on individual flow QoS specifications. 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 QoS specification can be associated with the group of flows.

We are currently investigating use of a symmetrically opposite specification to support receiver-based temporal sharing. In this case, flow-setup proceeds in the opposite direction from the receiver to the senders. Interaction between sender-based and receiver-based sharing is area for future research.

Note that there are alternative ways of specifying temporal sharing. For example, both the specification and the computation of aggregate QoS can be simplified by using a threshold on the number of flows, beyond which the aggregate QoS is used. However, this specification is quite restrictive. A less restrictive, and yet reasonably simple specification might be to use a range of thresholds and associated aggregate QoS specifications. The Beagle temporal sharing specification provides the most flexibility by allowing any combination of sharing to be specified. This design decision was driven by the fact that a structured resource management architecture allows resource brokers to utilize this flexibility, and yet limit complexity, by using high-level knowledge of the application domain and how flows share links in the core of the mesh. Further research is needed to determine common types of application sharing behavior and appropriate specifications that are simple to implement and yet fully expressive.

Beagle also supports hierarchical resource management which, among other uses, can be used to support runtime enforcement of temporal sharing. The main requirement that flows from this is that the signaling protocol has to support the distribution of resource trees to the individual local resource managers. This information is specified to Beagle in a *hierarchical grouping tree* that expresses the hierarchical sharing for a virtual link. The leaf nodes of the trees represent flows or simple flow aggregates; their QoS requirements are expressed by their individual flowspecs. The interior nodes of the tree have generic QoS service types associated with them (e.g. WFQ or guaranteed bandwidth), and service-specific rules describe how child node flowspecs are aggregated into the parent node flowspec. Using this information, Beagle can generate individual link resource sub-trees. This potentially requires two transformations. First, the service types specified in the hierarchical grouping tree have to be mapped onto the input parameters for the specific hierarchical scheduler at each link. Second, the virtual link will often be mapped onto a multi-hop path in which not all flows will traverse all paths, e.g. there may be parallel paths. Based on mesh flows that share a particular physical link in the network, Beagle has to prune the hierarchical grouping tree to eliminate flows that do not exist at that link.

VI. MESH SETUP

This section discusses mesh setup algorithms, and evaluates their performance using combined experimental and queueing analysis.

A. Mesh Setup Algorithms

Recall that the virtual mesh consists of a set of flows, which are aggregated to varying degrees in the mesh core. If the mesh includes N flows and the core contains N_a aggregate flows, $N_a \leq N$. Consequently, the mesh can be allocated in one of two ways: 1) *flow-based setup* — where each flow in the mesh is set up independently, and resource sharing and aggregation within the core are achieved in a completely distributed fashion; and 2) *core-based setup* — where the mesh core flows are set up separately and in parallel with the "edge" flows. Core-based setup assumes some centralized knowledge of how flows are aggregated in the core and what their resource requirements are.

Both flow-based and core-based mesh setup algorithms use third-party flow setup as their basic mechanism. Third-party flow setup involves allocating resources between any two network nodes from a third node. Beagle uses a specific implementation of third-party flow setup, which is explained later on in this section. We explain the two mesh setup algorithms next.

A.1 Flow-based Mesh Setup

In flow-based mesh setup, all the flows are set up individually and independently of each other. The mesh creator initiates parallel third-party setups of all flows in the mesh. The mesh setup completes when all the flow setups complete. However, depending on the application, the success of mesh setup need not depend on all flows being set up successfully.

In flow-based mesh setup, every flow setup request message carries enough information to "rendezvous" appropriately with other mesh flows in the core. Aggregate resource allocation and setting of various resource sharing policies in the core occur in a completely distributed fashion. The total amount of protocol processing in the mesh core is the sum of individual flow processing times. Also, different levels of route and state aggregation have no impact on the total processing time of the virtual mesh. This is because the request message size remains the same and the number of admission control decisions in the core is independent of varying degrees of route and state aggregation.

A.2 Core-based Mesh Setup

Every virtual mesh has its N flows aggregated to a smaller number N_a within the mesh core. Each end-to-end flow can be thought of as comprising three segments: one aggregated segment through the core and two individual segments at the edges on either end. In core-based mesh setup, the creator initiates parallel third-party setups of all the core flows and edge flows (a total of $2N + N_a$ flows). The mesh is set up successfully when all of these flows are set up. In contrast to flow-based setup, failures of core flow setup have different implications on the success of the mesh setup, as opposed to the failure of edge flow setup. The failure of a core flow setup affects all the edge flows that are aggregated into the core flow.

Setup request messages for core flows and for edge flows differ in several ways. The request message for an edge flow is essentially the same as for an end-to-end flow — the only difference being that the edge flow request message does not carry extra information to rendezvous with other mesh flows. On the other hand, the setup request message for a core flow must contain information describing the aggregate traffic flow. Other differences depend on the extent of route and state aggregation within the core of the mesh.

Route aggregation primarily affects the size of the core flow request message. In the absence of route aggregation, a core flow may have to be split within the core into several flows that take alternate paths through the core due to a lack of sufficient resources for the core flow. This requires extra information in the core setup message, in order to calculate the QoS requirements of the resulting core flows.

State aggregation, on the other hand, primarily affects the number of admission control decisions in the core. Consider hierarchical scheduling as an example. With state aggregation, a core flow setup generates a single node in the link resource tree. This requires a single admission control decision to be made in local resource managers. Without state aggregation, a core flow setup creates an entire sub-tree in the link resource tree. This requires admission control decisions involving not only the aggregate resource requirement, but also the distribution of individual resources within the sub-tree.

Thus, in core-based mesh setup, both route and state aggregation can reduce processing time within the core. They also reduce signaling load in the mesh core, due to a smaller number of core flows as compared to mesh flows. A third factor also influences performance in core-based setup: parallelism of mesh setup. Because core flows and edge flows are all established in parallel, the average length of the flow segments is reduced, leading to a decrease in mesh setup delay. The advantages of a parallel flow setup architecture are described in [9]. The effects of these three factors are evaluated in greater detail in the next section.

B. *Performance Evaluation*

In this section we present a performance evaluation of the two mesh setup algorithms. We use *virtual mesh setup time* as the performance metric by which we compare the two mesh setup approaches. In comparing the performance of edge-based and core-based setup, we evaluate two extremes of a spectrum which is defined by the extent of aggregation within the core. Edge-based mesh setup has no aggregation in the core and represents one extreme of the spectrum. For the core-based case, we assume that flows stay aggregated (both state aggregation and route aggregation) throughout the mesh core. This is the best case for core-based mesh setup and represents the other extreme of the spectrum. Performance of core-based mesh setup falls between the two extremes depending on how flows are aggregated within the core.

Since third-party initiated flow setup mechanism is the fundamental mechanism to both core-based and flow-based mesh setup, we present the implementation of third-party flow setup in Beagle next, followed by a performance analysis of the two mesh setup algorithms.

B.1 Third-party Flow Setup

We now describe one particular way to implement third-party initiated flow setup. The mechanism we describe is implemented in Beagle. 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 neighboring 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 it directly to A. A strips off the PRXOY_REQ container and forwards the message toward 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

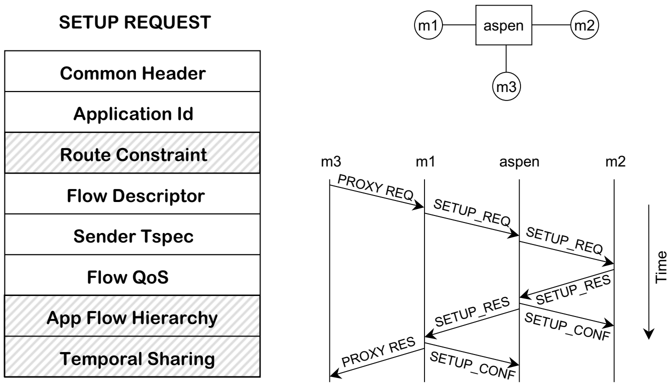

Fig. 4. (a) Format of the SETUP REQUEST message, (b) Message flow for flow setup initiated by a third party.

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 the 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 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`.

Flow-based mesh setup uses the same SETUP_REQ message as shown in Figure 4(a). Core-based flow setup, however, uses a modified form of the SETUP_REQ message containing a list of Flow Descriptor objects, describing the aggregate traffic flow. Depending on state and route aggregation, other modifications include carrying a list of Sender Tspec and Flow QoS objects, describing the individual QoS requirements of the flows aggregated into the core flow.

Third-party flow setup in Beagle as presented above requires the use of hard state. Clearly, there are alternate implementations. One of the alternatives is to use soft state with periodic refreshes. While soft state makes it easier to deal with error conditions, performance could be worse because periodic refreshes add to the signaling load on routers.

B.2 Experimental Evaluation

We measured the performance of the Beagle prototype implementation to determine the breakdown of flow processing time into several components such as route lookup, admission control, message parsing etc. The experiment involved repeated trials of setting up flows on the Darwin testbed [4]. The Darwin testbed consists of Pentium II 266 MHz PC routers. During each trial, we measured the performance of the prototype using the high-resolution Pentium cycle counter. The breakdown of the total processing time is shown in Table I. A large fraction of the total processing time (70%) is spent in interacting with the local resource manager (LRM).

TABLE I

BREAKDOWN OF FLOW SETUP PROCESSING TIME IN MICROSECONDS AT A ROUTER.

Time	Function	Value (μs)
T_{REQ}	SETUP_REQ processing	336
T_{RES}	SETUP_RES processing	146
T_{CONF}	SETUP_CONF processing	126
T_{ROUTE}	Route lookup	119
T_{LRM}	LRM interaction	1578

The total flow setup processing time in a router is the sum of all the components shown in Table I. The total processing time can be broken up into three components t_1, t_2 and t_3: each component representing the set of processing steps that are triggered by the arrival of SETUP_REQ, SETUP_RES and SETUP_CONF messages. The three component times are given by:

$$t_1 = T_{REQ} + T_{ROUTE} + T_{LRM} = 2033\mu s \quad (1)$$

$$t_2 = T_{RES} = 146\mu s \quad (2)$$

$$t_3 = T_{CONF} = 126\mu s \quad (3)$$

These component times determine the service time distribution for the queueing model described below.

B.3 Queueing Network Model

Because mesh flows are set up in parallel, the mesh setup time is the maximum of flow setup times. However, in evaluating the virtual mesh setup time, for tractability reasons, we assume that the virtual mesh setup time is determined by the flow setup time of the flow which traverses the maximum number of hops R. Therefore in comparing the performance of mesh-based and core-based setup, we compare the flow setup time for the maximum length flow (in number of hops R). This is an optimistic assumption that is accurate at light loads, but not accurate at very high loads. At high loads this assumption underestimates the virtual mesh setup time.

We use a queueing network model to determine the end-to-end flow setup time. Each router is represented as a node in the queueing network. The queueing delay at each node is calculated by using a decomposition assumption that treats each node as an independent M/G/1 queue, i.e the flow arrival process at each router is assumed to be an independent Poisson process. With this assumption, and using the service time distribution constructed by using the processing times measured in the previous section, the queueing delay at each node is given by:

$$f(\rho, t_1, t_2, t_3) = \frac{1}{3}(t_1 + t_2 + t_3) + \frac{\rho(t_1^2 + t_2^2 + t_3^2)}{2(1-\rho)(t_1 + t_2 + t_3)} \quad (4)$$

where ρ is the signaling load at the router in the range (0, 1).

In flow-based mesh setup, if we assume the maximum length of a flow in the mesh is R (in number of routers) and the signaling load on each router is the same, the average mesh setup time for the flow-based case is give by:

$$T_{flow-based} = 3Rf(\rho, t_1, t_2, t_3) \tag{5}$$

where the factor 3 appears because a flow setup requires all three components t_1, t_2 and t_3.

In core-based mesh setup, as mentioned above, three factors determine the performance of core flow setup: 1) state aggregation, 2) route aggregation and 3) parallel setup of shorter flow segments. We introduce three parameters a, x and s to capture the effects of these factors in the queueing model. Parameter $a = N/N_a$ is the aggregation ratio, which determines to what extent flows are aggregated within the mesh. It lies in the range $(1, N)$, with higher values indicating greater aggregation. This parameter determines the signaling load in the core. Parameter x is the ratio of the processing times for a core SETUP_REQ message to a edge SETUP_REQ message. Higher values of x indicate a larger penalty for processing core flow SETUP_REQ message, mainly due to lists of objects carried in the core flow setup case. Parameter s identifies what fraction of the mesh lies within the core, and is a measure of the extent of parallelism that can be exploited in the core-based setup. s is a fraction in the range (0, 1) and if there are R routers in the maximal length flow, then there are sR routers in the core and $(1-s)R$ routers in the edges. Using these assumptions, the mesh setup time in the core-based case can be written as follows:

$$t_{core-based} = \max(t_{edge}, t_{core}, t_{edge}) \tag{6}$$

where t_{edge} and t_{core} are random variables representing the edge and core components of the mesh setup delay and are given by,

$$t_{edge} = \sum_{i=1}^{(1-s)R/2} t_{edge,i} \tag{7}$$

$$t_{core} = \sum_{i=1}^{sR} t_{core,i} \tag{8}$$

where $t_{edge,i}$ and $t_{core,i}$ are random variables representing the per-node setup delays at the edge and core routers respectively.

We use the technique described in [10] to calculate the average mesh setup delay in the core-based case by modeling the queueing network as an activity graph. The activity graph has three parallel queuing networks representing the nodes in the edge and core setup portions of the mesh. The delay distribution at each node (of random variables $t_{edge,i}$ and $t_{core,i}$) is obtained by numerically inverting the Laplace transform of the delay distribution given by,

$$F^{\star}_{edge}(s) = \frac{(\mathrm{e}^{-st_1} + \mathrm{e}^{-st_2} + \mathrm{e}^{-st_3})s(1-\rho)}{3(s-\lambda) + \lambda(\mathrm{e}^{-st_1} + \mathrm{e}^{-st_2} + \mathrm{e}^{-st_3})} \tag{9}$$

$$F^{\star}_{core}(s) = \frac{(\mathrm{e}^{-sxt_1} + \mathrm{e}^{-st_2} + \mathrm{e}^{-st_3})s(1-\rho')}{3(s-\lambda') + \lambda'(\mathrm{e}^{-sxt_1} + \mathrm{e}^{-st_2} + \mathrm{e}^{-st_3})} \tag{10}$$

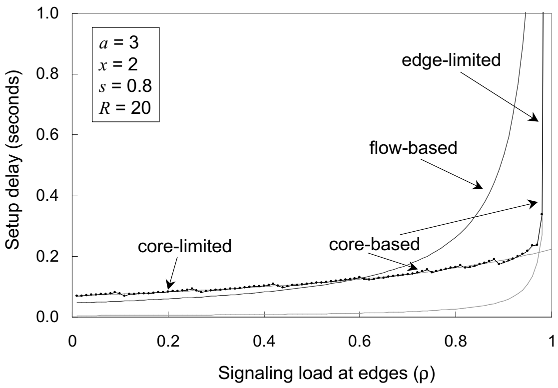

Fig. 5. Contributions of core-limited and edge-limited segments to overall setup delay in the core-based case.

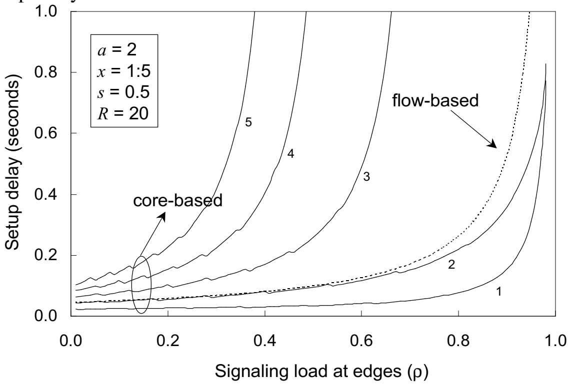

Fig. 6. Effect of varying the processing ratio x on the mesh setup time.

where, $\rho' = \frac{xt_1+t_2+t_3}{a(t_1+t_2+t_3)}\rho$, $\lambda = \frac{3\rho}{t_1+t_2+t_3}$ and $\lambda' = \frac{\lambda}{a}$. The individual node delay distributions are then approximated by discrete lattice distributions which are then combined to obtain the end-to-end delay distribution. See [10] for details of the procedure.

B.4 Results

In this section we compare the performance of core-based and flow-based mesh setup algorithms and study the effect of the three parameters a, x and s on performance. We use the queueing model and experimental results presented in the previous sections, and generate delay-throughput curves for the flow-based and core-based cases. In all the results presented below, we assume that the maximal length flow is through 20 routers, i.e., $R = 20$. Each graph plots mesh setup delay versus the signaling load at the edges ρ.

Figure 5 plots virtual mesh setup delay for the flow-based and core-based cases with an aggregation factor of 3, processing ratio of 2, and a core span of 0.8. For flow-based setup, the setup delay curve has the form of a standard delay-throughput curve for an M/G/1 queue. Setup delays increase exponentially as the signaling load approaches unity. On the other hand, in the core-based setup, the setup delay curve has two clear regions. At light loads, the setup delay is dominated by the delay in the core. This is because of the extra processing needed to

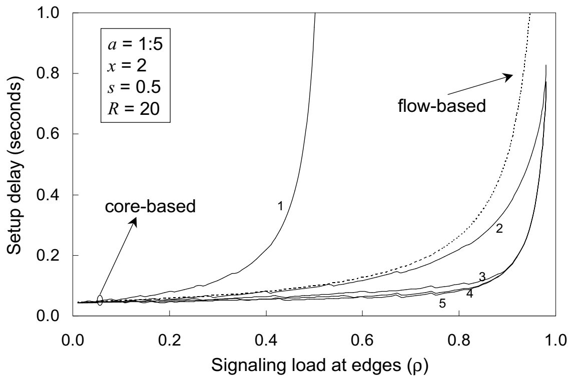

Fig. 7. Effect of varying the aggregation ratio a on the mesh setup time.

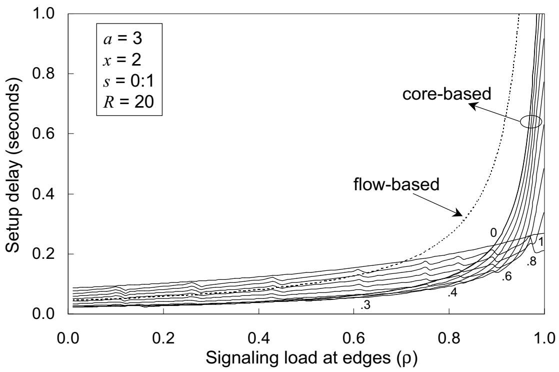

Fig. 9. Effect of varying the core-span factor s on the mesh setup time.

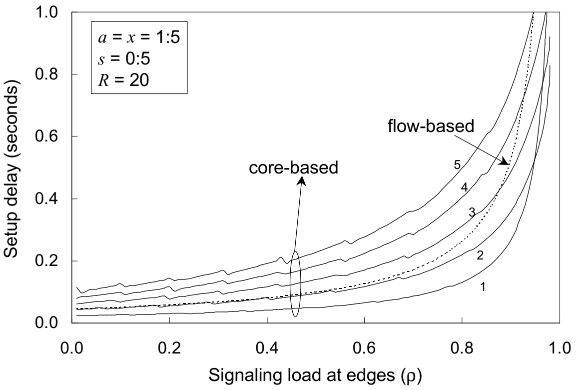

Fig. 8. Effect of varying the aggregation factor a and processing ratio x together.

setup a core flow. At heavy loads, the setup delay is dominated by the queueing delay in the edges. This is because aggregation reduces the load in the core, and queueing delay in the core is negligible compared to the edges. This shows that at light loads, flow-based setup can outperform core-based setup depending on the processing factor x. At heavy loads, core-based setup is better than flow-based setup because it sets up shorter flow segments in parallel. The slight oscillatory behavior in the core-limited portion of the graph is purely a modeling artifact caused by using a Fourier series approximation to calculate the inverse Laplace transforms.

Figure 6 shows the effect on mesh setup delay of varying the processing ratio x. As expected, the figure shows that increasing the processing ratio increases the delay in the core-limited portion of the graph for core-based setup. Interestingly, it can be seen that as long the processing ratio is less than or equal to the aggregation factor, core-based setup performs better than flow-based setup. If the processing ratio is higher than the aggregation factor, the performance of core-based setup is much worse. This is because the signaling load factor in the core is higher than the edge load factor when x is greater than a.

In Figure 7, we examine the effect of varying the aggregation ratio a. Similar to the previous result, we see that core-based setup performs poorly compared to flow-based setup when the aggregation ratio is lower than the processing factor. Higher values of the aggregation ratio give better performance, although the improvement beyond $a = 3$ is not significant.

The aggregation factor and the processing ratio have opposite effects on the performance of core-based setup. To study their interaction, we vary both a and x together in Figure 8. As seen from the figure, at light to moderate loads, the effect of the processing ratio dominates. At very high loads, the aggregation factor comes into play, as can be seen by the overlapping curves for the core-based setup. Finally, Figure 9 shows the effect of varying the core-span factor s. The core-span factor determines the amount of parallelism that can be exploited in the core-based setup case. As seen from the figure, the best performance occurs between $s = 0.3$ and $s = 0.4$. This is as expected since the maximum parallelism occurs at $s = 0.33$.

Overall, the results demonstrate that state and route aggregation affects the performance of core-based setup much more than the parallel setup of the core and edge segments. Core-based setup performs better or worse than flow-based setup depending on whether the processing factor x is less than or greater than the aggregation ratio a. Even when they are equal, the processing factor has the dominant effect on performance. Structured applications, as described earlier, are likely to have both state and route aggregation. For these applications, it is likely that the aggregation ratio can be much higher than the processing factor. Therefore, these applications can benefit by using core-based mesh setup.

VII. BEAGLE

Figure 10 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`). Applications and/or resource brokers (Xena) can invoke Beagle by making appropriate Beagle interface function calls. The Beagle API (shown by the shaded portions in Figure 10) is a library that is compiled in as part of the application. It communicates with the Beagle daemon through UNIX domain sockets. Beagle API calls support the creation of meshes and flows

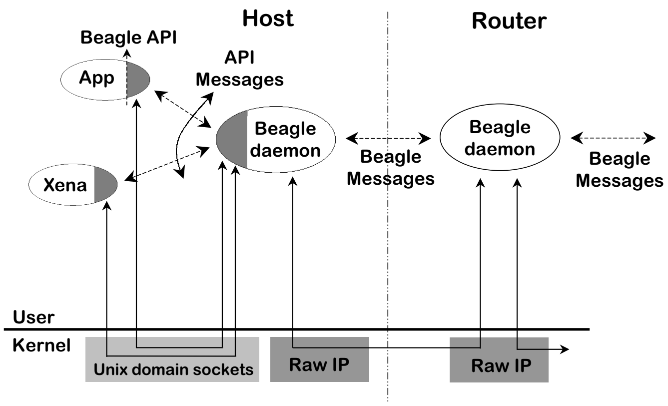

Fig. 10. Beagle prototype and Beagle API implementation.

and also the allocation of computation and storage resources. The Beagle API also has calls to attach to an existing mesh and to dynamically modify flow characteristics during runtime. The Beagle API also has a callback mechanism that is used to asynchronously notify applications about various events such as setup success/failure, incoming requests, etc. 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 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 [11] or ATM PNNI [3]. 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. The issues concerning appropriate mechanisms to fully recover application mesh state are still being explored. It is expected that a combination of soft-state and hard-state approaches might work well for recovering different parts of mesh state.

VIII. Related Work

The development of QoS-capable networks like ATM and Integrated Services Internet has triggered the development of resource allocation protocols capable of supporting applications that request per-flow QoS. In the context of the Integrated Services Internet, RSVP [2] and SCMP [12] 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 a unicast session, an entire multicast session, or a sender-specific multicast flow within a multicast session. RSVP uses soft state with periodic refreshes to establish reservation state. By merging reservation requests from downstream receivers, RSVP has some core-based setup properties. But the core arises in an unstructured way and is limited to a multicast group. 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 [3] protocol was developed to set up 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. Among these are the Tenet protocols [13] [14], CMAP/CNMP [15] and DCPA [16]. 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 supports these applications well by meeting the requirements listed in Section III. Beagle supports both receiver-oriented and sender-oriented resource allocation.

The recent development of differential services architectures [17] [18] has brought to forefront the need to aggregate resource allocation in the core of the Internet for better scalability. Several protocols support aggregation to varying degrees. Recently, extensions have been proposed to RSVP [19] [20] [21] that advocate the use of CIDR style filters to support aggregation, and provide 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 been addressed in RSVP [2] and Tenet-2 [6]. 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. Resource sharing in Tenet-2 is achieved by defining groups of flows and associating an aggregate flowspec with the group. The group flowspec is used whenever the number of flows sharing a link exceeds a threshold. As mentioned before, both approaches address only the sender-based resource sharing model. Beagle generalizes support for resource sharing by supporting receiver-based sharing as well. Beagle also provides support for runtime enforcement mechanisms such as hierarchical scheduling. Sharing is not restricted to flows within a multicast session.

The performance and complexity of a signaling protocol has also received much attention lately. Parallel flow setup approaches [9] have been proposed to improve signaling performance. The YESSIR signaling protocol [22] focuses on reducing the complexity and improving the scalability of RSVP. It also provides support for sender-based resource sharing and "partial" reservations, where an end-to-end flow can be composed of reserved and best-effort segments. Beagle focuses on mesh setup performance and uses a parallel setup algorithm similar to that used in [9] to improve signaling performance. Use of a structured resource management architecture and support for mixed-reservations in Beagle generalizes the notion of partial reservations supported by YESSIR.

IX. Conclusions

This paper outlined the new requirements placed on signaling protocols by emerging multiparty, multiflow applications, discussed how these requirements might be supported, and presented the design of the Beagle signaling protocol that meets the requirements. We showed that for these structured applications, there is an advantage in adopting an integrated approach towards resource management. The paper makes several contributions in this area. The virtual mesh concept is presented as a key abstraction through which signaling protocols can support the new requirements placed by structured applications. We presented the core-based approach to mesh setup and showed that it can improve performance over flow-based mesh setup in some cases. We argued that signaling protocols should support the receiver-based resource sharing model in addition to sender-based sharing, and also provide support for enforcement mechanisms such as hierarchical scheduling.

The paper also presented the design and implementation of the Beagle signaling protocol where most of these ideas have been realized. A prototype implementation of Beagle runs on the Darwin testbed [4]. While 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. Better and more intuitive representation of sharing behavior needs to be explored.

References

[1] 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*, Napa, May 1998, IEEE, pp. 187–195.

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

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

[4] 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.

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

[6] Amit Gupta, Winnie Howe, Mark Moran, and Quyen Nguyen, "Resource sharing in multi-party realtime communication," in *Proceedings of INFOCOM 95*, Boston, MA, Apr. 1995.

[7] S. Deering, "Host extensions for IP multicasting," August 1989, RFC 1112.

[8] 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*, Cannes, September 1997, ACM, pp. 249–262.

[9] M. Veeraraghavan, G. L. Choudhury, and M. Kshirsagar, "Implementation and analysis of parallel connection control (pcc)," in *Proceedings of IEEE Infocom '97*, Kobe, Japan, April 1997, IEEE.

[10] G. L. Choudhury and D. Houck, "Combined queueing and activity network based modeling of sojourn time distribtuions in distributed telecommunications systems," in *Proceedings of 14th International Congress*, Antibes Juan-les-Pins, France, June 1994, IEEE, pp. 525–534.

[11] 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.

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

[13] 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*, vol. 4, no. 1, pp. 1–10, February 1996.

[14] 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*, Durham, New Hampshire, Apr. 1995, pp. 255–266.

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

[16] M. Veeraraghavan and T. F. La Porta, "Object-oriented analysis of signaling and control in broadband networks," *International Journal of Communication Systems*, vol. 7, no. 2, pp. 131–147, April 1994.

[17] 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.

[18] 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.

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

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

[21] 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.

[22] Ping Pan and Henning Schulzrinne, "Yessir: A simple reservation mechanism for the internet," in *Proceedings of the 8th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 98)*, Cambridge, United Kingdom, July 1998.