Extensible Signaling for Temporal Resource Sharing

Prashant Chandra, Peter Steenkiste, Allan Fisher

Abstract— **The Internet is rapidly evolving from a network that provides basic best-effort communication service, to an infrastructure capable of supporting complex value-added services. These services typically have multiple flows with inter-dependent resource requirements. Taking advantage of these relationships, the same set of resources can be shared among multiple service flows over time leading to significant resource gains. We call this type of sharing *temporal resource sharing*. Exploiting temporal sharing requires support in the signaling protocol that performs resource allocation for the related flows. However, the signaling protocols previously described in literature provide very limited support for such temporal sharing.**

In this paper, we examine the problem of supporting temporal sharing in a signaling protocol. This paper makes the case that temporal sharing support must be designed to be *extensible*, so that service providers can define and implement new sharing behaviors without having to modify the signaling protocol. We motivate the need for an extensible design by showing that the range of possible temporal sharing behaviors is large and supporting the most general forms of temporal sharing is computationally expensive. We then present a design for extensible signaling support for temporal sharing.

We have implemented the temporal sharing design presented in this paper in the *Beagle* signaling protocol. We present an evaluation of the Beagle design and contrast it with other signaling protocols like RSVP and Tenet-2.

Keywords— **Resource allocation, Signaling protocols, Active networks.**

I. Introduction

THE Internet is rapidly transitioning from a set of wires and switches that carry packets to a sophisticated infrastructure that delivers a set of complex value-added services to end users. These services typically involve multiple end-points and use multiple flows often with inter-dependent resource requirements. Traditional flow-based signaling protocols allocate resources for each flow independently. This is based on the underlying assumption that each flow in the network is independent of all other flows in terms of its resource utilization. However, most services with multiple flows exhibit temporal relationships in the way their flows utilize the resources allocated to them. In such cases, these "related" flows can share the same set of resources over time. We call this type of behavior *temporal sharing* and define it as *the sharing of resources among multiple flows with temporally interleaved resource usage*. Temporal sharing forms a middle ground between independent flow-based allocation and periodic renegotiation by combining the low signaling overhead and predictable behavior of independent flow based allocation, with savings in resource consumption obtained using periodic renegotiation.

Temporal sharing was first introduced in the original RSVP design paper [1]. RSVP introduced the notion of resource reservation styles that allowed different senders to a multicast group to share the same set of resources. A subset of the styles introduced in the original paper is supported in the RSVP specification [2]. Temporal sharing has also been studied in the context of other signaling protocols like Tenet-2 [3] and ST2+ [4]. Although these signaling protocols represent an important first step in exploiting temporal sharing, the "one size fits all" approach they take limits their usefulness. The support they provide is mostly suited for conference style applications. However, as will be described later, there is a much wider spectrum of application behaviors that can benefit by using temporal sharing. The temporal sharing support provided by the above protocols is inadequate for these applications.

This paper makes the case that temporal sharing support must be designed to be *extensible*, so that service providers can define and implement new sharing behaviors without having to modify the signaling protocol. This is based on the observation that temporal sharing is an optimization that closely depends on the behavior of the service and is therefore best performed using service-specific knowledge. We consider the range of possible temporal sharing behaviors and show that while supporting the most general forms of temporal sharing is computationally expensive, several useful temporal sharing behaviors can be supported cheaply either by using service-specific knowledge, or by trading off resource efficiency for computation overhead. We describe the design and prototype implementation of extensible temporal sharing support in a signaling protocol called *Beagle*.

The rest of the paper is organized as follows. Section II outlines the motivation for this work and gives examples of application behaviors that exhibit temporal sharing. In Section III we consider the range of possible temporal sharing behaviors. Section IV evaluates the computational complexity of supporting different temporal sharing behaviors. We discuss the design of extensible temporal sharing support in Section V. Section VI describes the Beagle prototype implementation. Section VII presents an evaluation of the Beagle implementation of temporal sharing. Finally, Section VIII contrasts our approach with related work and Section IX presents the conclusions.

II. Motivation

Most applications with multiple flows exhibit some form of temporal sharing. In this section we consider a few applications and show how temporal sharing can be exploited to save resources. We also the present the abstraction of a *flow group* that captures temporal sharing relationships in a general fashion.

A. Conferencing

The most well-known style of temporal sharing is that exhibited by conferencing applications with some form of

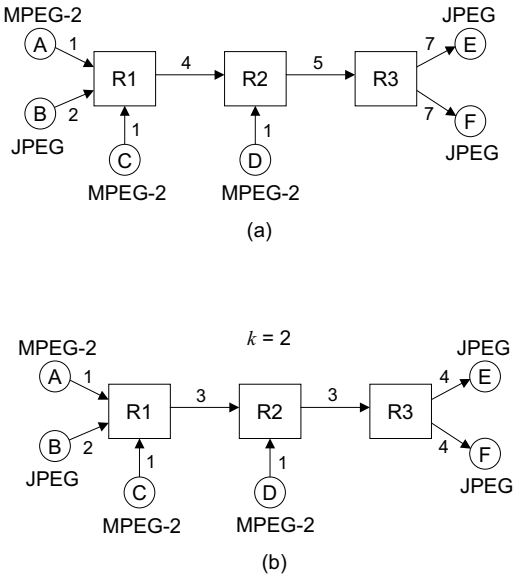

Fig. 1. Example of Beagle temporal sharing in a conference application with at-most two active speakers: (a) without temporal sharing, (b) with temporal sharing.

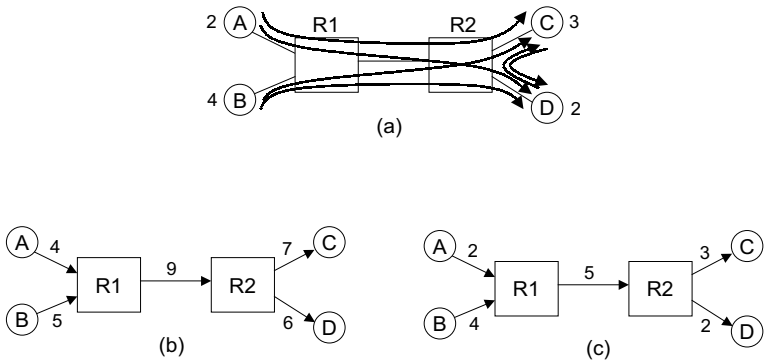

Fig. 2. Example of Beagle temporal sharing in a VPN application: (a) VPN topology, (b) without temporal sharing, (c) with hose-based VPN sharing.

"floor control" which limits the number of active speakers. Figure 1 shows the use of conference style temporal sharing in a video conference among six participants, A through F. Each participant multicasts video coded as either JPEG (requiring 2 units of bandwidth) or MPEG-2 (requiring 1 unit of bandwidth). The video conference application is structured so that there are at most two simultaneously active speakers. Figure 1(a) shows the resource allocation at links in the network with independent per-flow allocation. For simplicity, resource allocation is shown only along the directions specified on the links. Without temporal sharing, the resource allocation on a link increases with the increasing number of sources upstream. Therefore at receivers E and F a bandwidth of 7 units is allocated to account for video flows of all five sources upstream (two JPEG and three MPEG-2). Using conference style sharing all flows through a link share the same set of resources. As shown in Figure 1(b), it is sufficient to allocate bandwidth at each link that covers the total bandwidth of the two highest bandwidth sources upstream. For example, at receivers E and F, a bandwidth of 4 units is allocated to account for the two JPEG sources upstream. As shown in the figure, this significantly reduces the resource requirements when compared to the earlier scenario without temporal sharing.

Conferencing is an example where temporal sharing arises as an inherent property of the application. We call such applications *self-limiting applications* using the terminology from [5]. Other examples of self-limiting applications are distributed interactive simulations, multiparty games and statistical multiplexing.

B. Virtual Private Networks

Virtual Private Networks (VPNs) are overlay networks laid over the existing Internet that connect several sites together. An important component of a VPN service is resource management. Recently, a new model has been proposed for resource management in VPNs called the *hose model* [6]. According to the hose-model, in a VPN with N sites, each site i is connected by an access link of bandwidth h_i called a "hose". Therefore, a hose limits the amount of traffic generated or received by the site.

The hose model provides an opportunity for temporal sharing in VPNs. Consider a VPN with N sites interconnected together. Because each site can transmit to all the other sites, there are $N(N-1)$ unicast flows. Each unicast flow connecting a pair of sites is capable of using the full bandwidth of the hoses at each end. However according to the hose model, all the N flows generated at a site i share an aggregate bandwidth limit of h_i which is the capacity of the source hose at i. Similarly all the N flows destined to the site j share an aggregate bandwidth limit of h_j which is the capacity of the destination hose at j. Taking advantage of these limits imposed by the hoses, it is sufficient to allocate at each link in the network the bandwidth given by the minimum of total upstream and downstream hose bandwidths considering all the VPN flows at that link. This can lead to large resources savings over independent allocation.

Figure 2 shows the use of the hose-based VPN sharing style. We consider a VPN involving four sites A through D. A and D have hoses with 2 units of bandwidth, C has a hose with 3 units of bandwidth and B has a hose with 4 units of bandwidth as shown in the figure. Each site has a unicast flow from itself to each of the other sites. For simplicity, we only show all the unicast flows destined to sites C and D in Figure 2(a). Each unicast flow has a bandwidth requirement that is defined by the minimum of the two hose bandwidths at either end of the flow. For example, the flow from A to C would have a bandwidth requirement of 2 units. Figure 2(b) shows the bandwidth allocation with independent flow-based allocation. As with the previous example, we show bandwidth allocations only in the directions specified along the links to keep the figure simple. Figure 2(c) shows the bandwidth allocations with hose-based VPN sharing. As discussed before, at each link the bandwidth allocation is calculated by taking the minimum

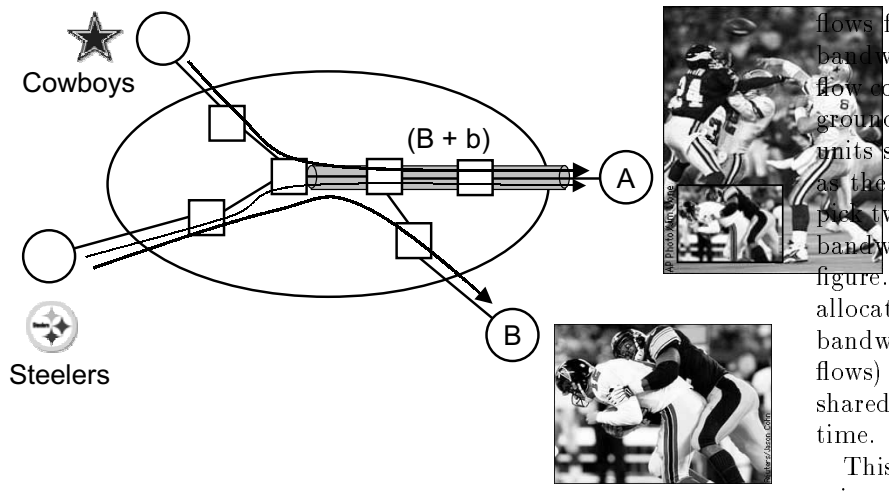

Fig. 3. Example of a broadcast TV application with picture-in-picture capability.

of the total upstream and downstream hose bandwidths at that link. For example, at the link from R1 to R2, the total upstream hose bandwidth is 6 units (A and B) and the total downstream hose bandwidth is 5 units (C and D). Therefore 5 units of bandwidth is allocated. As shown in the figure, there is significant reduction in bandwidth allocation at internal links (from R1 to R2) when compared to independent allocation. Also, the allocation at each hose reflects the limitation of that hose, in contrast to independent allocation. This shows that using hose-based VPN style sharing can significantly reduce the resource requirements of a VPN.

The hose-based VPN resource allocation is an example where temporal sharing arises because the network imposes limits on aggregate traffic carried by a group of flows. We call such applications *network-limited applications*. A more general example of a network-limited application is the *virtual mesh* as described in [7]. A virtual mesh is characterized by several end-points, a few designated routers and several virtual links between them. The virtual links might impose limits on aggregate resource usage which provides opportunities for temporal sharing. For example a virtual mesh over a Differential Services Internet [8], might have limits imposed on aggregate traffic at interconnection points between different administrative clouds.

C. Broadcast TV with Picture-in-Picture

Consider a broadcast TV application with a picture-in-picture (PIP) capability as shown in Figure 3. A typical scenario in which the PIP capability is useful is the live transmission of simultaneous sports events (e.g NFL football games). In the application shown in the figure, each football game is a separate video channel that is multicast to a set of independent viewers. Viewer A is watching two video channels simultaneously switching back and forth between the two channels making one channel the main channel and the other the inset. Viewer B, on the other hand, is tuned only into one of the channels.

One way to implement this application is to have two flows for each video channel. One flow would be a high bandwidth (B units shown using a thick line), high quality flow corresponding to the channel being viewed as the foreground and; the second flow would be a low bandwidth (b units shown using a thin line), low quality flow to be used as the inset. A viewer watching two channels would then pick two out of the four flows at a time, choosing one high-bandwidth and one low-bandwidth flow as shown in the figure. Temporal sharing can be exploited in this case by allocating resources for one high-bandwidth and one low-bandwidth flow (instead of allocating resources for all four flows) and allowing the viewer to dynamically associate the shared resources with the appropriate flows at any point in time.

This application is an example where temporal sharing arises because receivers tune into and out of a set of independent multicast flows over time. We call such applications *channel switching applications*, again using the terminology given in [5]. Other examples of channel-switching applications include distributed processing of data for an array of sensors (e.g. an array of radars or weather satellites) and Enhanced TV where a video flow is augmented by a real-time data flow carrying extra information about the video (e.g. real-time statistics during a televised football game).

D. Flow Groups

Temporal sharing between a group of flows can in general be represented by a *flow group*, which identifies the group of flows, and a rule to compute the resource requirements of the group of flows. The rule consists of a formula to calculate the resource requirements and a set of parameters (e.g. flow specs). The formula wil depend on the sharing behavior between the flows while the parameters will depend on the specific nature of this flow group instance.

Flows can be members of one or more flow groups. In the case of self-limiting applications, each flow is a member of one flow group. The group QoS spec associated with that flow group reflects the inherent limit imposed by the application. In the case of network-limited applications, each flow is a member of m flow groups where m is the number of network-imposed aggregate limits encountered along the path of the flow. For example, $m = 2$ in the case of hose-based VPNs because each end-to-end flow between any two sites goes through one source hose and one destination hose. Finally, in the case of channel-switching applications, each flow is typically a member of more than one flow group where, a flow group is defined by the receiver.

III. Range of Temporal Sharing Behaviors

The examples presented in the previous section represent a small sample of a wide range of possible temporal sharing behaviors. In this section we explore the range of possible temporal sharing behaviors and present a two-dimensional design space that characterizes all possible temporal sharing behaviors. In the next section, we evaluate the computational complexity of supporting different temporal shar-

ing behaviors.

A. Flow Types

The earlier examples exhibit two types of flows, *related flows* and *independent flows*, that differ in how the aggregate resource requirement is calculated and how the shared resource is arbitrated among the flows during runtime.

Related flows are those that exhibit temporal relationships among sources. In this case temporal sharing occurs as a result of the peaks of activity of different sources being interleaved in time. This can occur as inherent application behavior as in the case of self-limiting applications, or can be artificially imposed to save resources as in the case of network-limited applications. For related flows, the shared resource is arbitrated in an end-to-end fashion through coordination among the sources. This coordination can be achieved in several ways: for e.g., at the application layer in the form of a conference manager or at the transport layer using protocols such as TCP to detect the available bandwidth. The algorithm to calculate the aggregate resource requirement for related flows has to determine the set of flow groups that minimizes the total resource requirement at a link. Temporal sharing for related flows characterizes source behavior and therefore applies to both unicast and multicast flows.

Independent flows are those where, the sources do not exhibit any temporal relationships. Temporal sharing is still possible in this case because receivers switch over time among a set of sources from which they receive data. This type of temporal sharing is exhibited by channel-switching applications where, a receiver need not allocate resources for all sources in which it is interested. Instead, it allocates an aggregate set of resources enough to handle the worst-case combination of simultaneous sources, and then switches among the sources at runtime. For independent flows, the shared resource is explicitly associated with the set of currently "active flows" specified by the receivers downstream. The calculation of the aggregate resource requirement for groups with independent flows has to determine the maximum possible resource requirement depending on the worst-case choices made by receivers downstream. Temporal sharing for independent flows characterizes receiver behavior and applies to multicast flows.

B. Temporal Sharing Design Space

The two types of flows described above capture all possible temporal sharing behaviors by representing both source and receiver relationships. Therefore, related and independent flow types define a design space for temporal sharing. All possible temporal sharing behaviors can be represented as points in this design space as shown in Figure 4. The two axes defining this space are the two flow types in the network. These two axes define the options for temporal sharing between groups of flows and applications can have multiple groups of flows with different sharing behavior

Each axis represents the tradeoff between computational complexity and resource efficiency. Applications can achieve the lowest resource consumption by specifying the most general forms of temporal sharing for the two types of flows. As shown in the next section, this is computationally expensive. Applications can either opt for less optimal resource consumption in favor of simplified specification as in the case of broadcast TV with picture-in-picture example; or use application-domain knowledge to simplify the calculation complexity, while still achieving best possible resource consumption as in the case of conferencing and VPN applications.

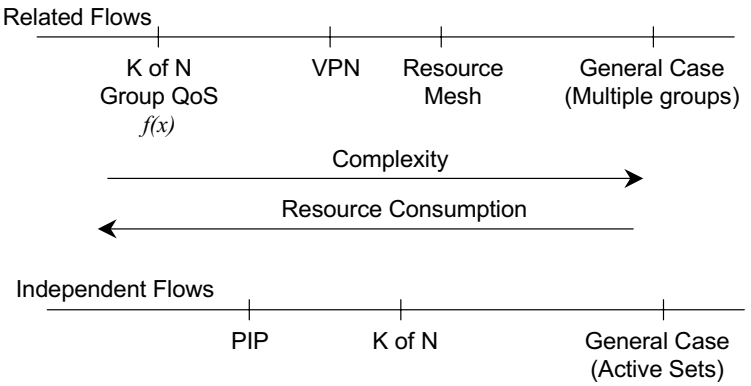

Fig. 4. Points in the design space of temporal sharing behaviors

IV. Analysis of Sharing Behaviors

In an ideal world, there would be a single general formula for specifying temporal sharing and the signaling protocol would support this single format. In this section we show that the general way of supporting temporal sharing is very expensive. Fortunately, by exploiting application properties (customization) or by being less aggressive in exploiting temporal sharing, the cost of temporal sharing can be reduced. In this section we illustrate this using the VPN sharing type as an example. A more detailed discussion on the complexity of exploiting temporal sharing can be found elsewhere [9].

A. General Case

Let F be the set of flows $f_i, i = 1 \cdots N$ at a link. Let $G_j, j = 1 \cdots M$ be the set of flow groups at that link.

$$G_j = \{f : f \in F\} \tag{1}$$

Hence $G_j \subseteq F$. In the most general form of temporal sharing for related flows, each flow f_i can be a member of any number of groups G_j. Define $\mathcal{F}$ to be the family of single subsets of F and $\mathcal{G}$ to be the family of subsets of F defined by the flow groups G_j.

$$\mathcal{F} = \{\{f_1\}, \{f_2\}, \cdots, \{f_N\}\} \tag{2}$$

$$\mathcal{G} = \{G_1, G_2, \cdots, G_M\} \tag{3}$$

Let $\mathcal{X} = \mathcal{F} \bigcup \mathcal{G}$. Each element i in $\mathcal{X}$ has a resource requirement q_i which is either the QoS of the flow f_i or calculated using the rule associated with the group G_i. The calculation of the aggregate resource requirement requires determining the family $\mathcal{C} \subset \mathcal{X}$ that covers the set F with

minimum resource requirement.

$$F = \bigcup_{S \in \mathcal{C}} S : \sum_{S \in \mathcal{C}} q_S \leq \sum_{S \in \mathcal{C}'} q_S, \mathcal{C}' \subseteq \mathcal{X}, \mathcal{C}' \neq \mathcal{C}$$

This problem is the well-known weighted set covering lem and is known to be NP-complete.

B. Virtual Private Networks

In the hose-based VPN case, each flow f is a m of two groups: one reflecting the limit on the agg traffic generated by the source site and another ref the limit on the aggregate traffic that can be sin the destination site. Therefore, in this case we have two families of flow groups $\mathcal{G}_\mathcal{S}$ and $\mathcal{G}_\mathcal{R}$ corresponding to flow groups defined by the source access link limitations and destination access link limitations respectively. And both families of flow groups consist of *disjoint* subsets that cover F:

$$F = \bigcup_{S \in \mathcal{G}_\mathcal{S}} S, \quad \text{and} \quad \forall S, S' \in \mathcal{G}_\mathcal{S}, S \cap S' = \phi \qquad (5)$$

$$F = \bigcup_{S \in \mathcal{G}_\mathcal{R}} S, \quad \text{and} \quad \forall S, S' \in \mathcal{G}_\mathcal{R}, S \cap S' = \phi \qquad (6)$$

A characteristic of the hose-based VPN application is that $\mathcal{G}_\mathcal{S}$ and $\mathcal{G}_\mathcal{R}$ are the only two coverings of the set F to consider because all other coverings contain one of these two and hence have a higher resource requirement. To state this more formally, let $\mathcal{G} = \mathcal{G}_\mathcal{S} \cup \mathcal{G}_\mathcal{R}$ represent the family of all flow groups at the link. If $\mathcal{C} \subseteq \mathcal{G}$ is any covering of the set F then, $\mathcal{C} \cap \mathcal{G}_\mathcal{S} = \mathcal{G}_\mathcal{S}$ or $\mathcal{C} \cap \mathcal{G}_\mathcal{R} = \mathcal{G}_\mathcal{R}$. To prove this, we represent the set of flows F at the link under consideration in a $p \times q$ matrix M where, p is the number of upstream sources and q is the number of downstream destinations. The family of groups $\mathcal{G}_\mathcal{S}$ and $\mathcal{G}_\mathcal{R}$ are formed by taking rows and columns of the matrix M respectively. All the elements of the matrix are covered by the set of rows $\mathcal{G}_\mathcal{S}$ or by the set of columns $\mathcal{G}_\mathcal{R}$. Any other attempt to cover all the elements of the matrix would involve a subset of the rows and all columns or vice-versa. Therefore any other covering of the set F would include either the source group family or the destination group family and has a higher resource requirement.

Because $\mathcal{G}_\mathcal{S}$ and $\mathcal{G}_\mathcal{R}$ contain disjoint subsets of F, the set covering problem reduces to an iteration among the groups G_i and with group G_i being chosen if its resource requirement is less than the sum of the member flows' resource requirements. The aggregate resource requirement q for the set of flows F can be calculated as follows:

$$q = \min \left\{ \sum_{S \in \mathcal{G}_\mathcal{S}} \min \left\{ q_S, \sum_{f \in S} q_f \right\}, \sum_{S \in \mathcal{G}_\mathcal{R}} \min \left\{ q_S, \sum_{f \in S} q_f \right\} \right\} \qquad (7)$$

The worst-case complexity of calculating the total resource requirements is of the order $O(N)$ where N is the number of flows at the link.

In summary, calculating the aggregate resource requirement for a set of related flows at a link can be NP-hard in the general case. But, using domain knowledge in the case of the VPN service, the calculation complexity can be reduced to be linear in the number of flows at the link. Note that this reduction in complexity is not achieved at the expense of resource consumption.

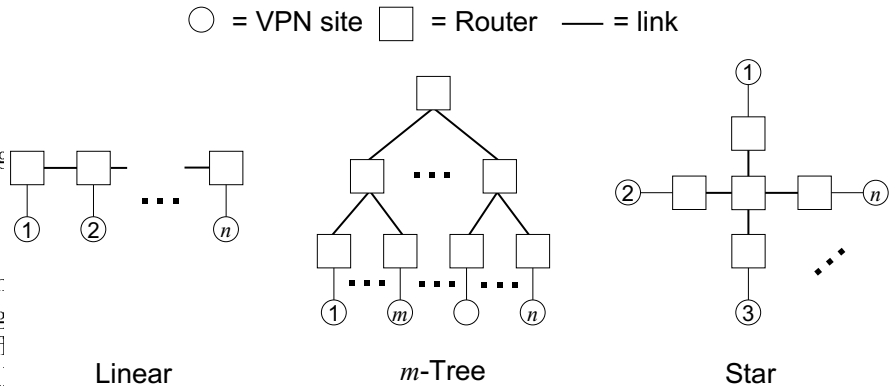

Fig. 5. Example network topologies for calculating asymptotic resource consumption gains.

TABLE I
RESOURCE ALLOCATION FOR A VPN SERVICE WITH n SITES, EACH WITH UNIT HOSE CAPACITY.

Topology	Independent	VPN sharing	Ratio
Linear	$\frac{n(n^2-1)}{3}$	$\frac{n^2}{2}$	$\frac{2}{3}(n - \frac{1}{n})$
m-Tree	$\frac{2n[(m-1)n\log_m n - n + 1]}{m-1}$	$2n \log_m n$	$n - \frac{n-1}{(m-1)\log_m}$
Star	$2n(n-1)$	$2n$	$n-1$

C. Resource Allocation Gain

In [5], the self-limiting and channel-switching temporal sharing styles are analyzed to calculate the resource allocation gains obtained over the three representative network topologies shown in Figure 5. Using the same methodology, we carry out an analysis of the VPN sharing style to determine the gains in resource allocation. Consider a VPN with n sites connected by one of the three candidate topologies. We assume that each site has unit hose capacity. In the case of independent flow-based allocation, each site is connected to every other site by a unicast flow. Therefore there are $n(n-1)$ flows each of which can transmit at full hose capacity (i.e 1). Therefore the total resource allocation for the VPN is $n(n-1)A$ where A is the average length of the path for each flow. Due to space considerations, we omit the details of the calculation of the average path length A and refer the reader to [5]. The results are summarized in Table I.

In the case of hose-based VPN, as shown in Section IV the resource requirement at each link is given by $\min\{n_S, n_R\}$ where n_S is the number of source hose groups and n_R is the number of receiver hose groups at the link. Calculating the total resource requirement in this case requires the evaluation of the min function for each of the three topologies. Again, we omit the details of the analy-

sis and refer the reader to [5] for an evaluation of the min function. The results are summarized in Table I. As seen from Table I, using temporal sharing in hose-based VPN service reduces the resource requirements by a factor $O(n)$ over independent flow-based allocation.

V. Extensible Temporal Sharing Design

In this section we consider the design of signaling support for temporal sharing in the context of service-oriented networks. The design of signaling support for temporal sharing must meet the conflicting goals of supporting the wide range of possible temporal sharing behaviors and at the same time providing the support in an efficient manner. Supporting the most general form of temporal sharing is computationally expensive. While this complexity can be reduced by exploiting service properties or by trading off some complexity for increased resource allocation, this results in a large number of sharing styles that must be supported. Moreover, it is also likely that implementation and user experiences will lead service providers to implement new sharing behaviors.

Therefore, signaling support for temporal sharing must not be designed with hard-wired sharing mechanisms. Rather, the signaling support for temporal sharing must be designed to be *extensible*. Extensibility provides service providers with the ability to dynamically define and use new sharing behaviors without having to modify the signaling protocol. Such a design has the advantage of being able to cover the wide range of behaviors and at the same time allowing service providers to use service-specific knowledge and make intelligent tradeoffs to improve computation efficiency.

In this section, we discuss the design of extensible support for temporal sharing in the Beagle signaling protocol. We first briefly describe the design of the Beagle signaling protocol and mechanisms for flow setup. We then describe how temporal sharing information is represented in Beagle. We then focus on a single node and describe the design of the temporal sharing execution environment and give examples to illustrate the setup of flows with temporal sharing. Finally, we give examples to show the application of the conference and VPN sharing styles.

A. Overall Design and Beagle Mechanisms

A flow setup in Beagle is based on the standard *three-way handshake* mechanism realized by the exchange of three messages (SETUP_REQUEST, SETUP_RESPONSE and SETUP_CONFIRM) between neighboring routers along the path of the flow. The SETUP_REQUEST message carries a list of objects including those that provide information about the traffic carried by the flow and the QoS requirements for that flow. The Beagle entity at each router along the path processes this information, allocates resources required by the flow and forwards it to the next hop. In addition to the basic objects listed above, a SETUP_REQUEST message may also carry temporal sharing information if the flow is part of a flow group. This information is carried in the form of a `TemporalSharing` object which is described in the next section. The information in the `TemporalSharing` object is interpreted by dynamically downloaded code modules that implement support for a particular style of temporal sharing (such as conference, VPN, etc.). These code modules execute inside a *temporal sharing execution environment (TSEE)* which is responsible for interacting with the Beagle entity at that router to setup resources for flows with temporal sharing behavior. The design of the TSEE is described in a later section.

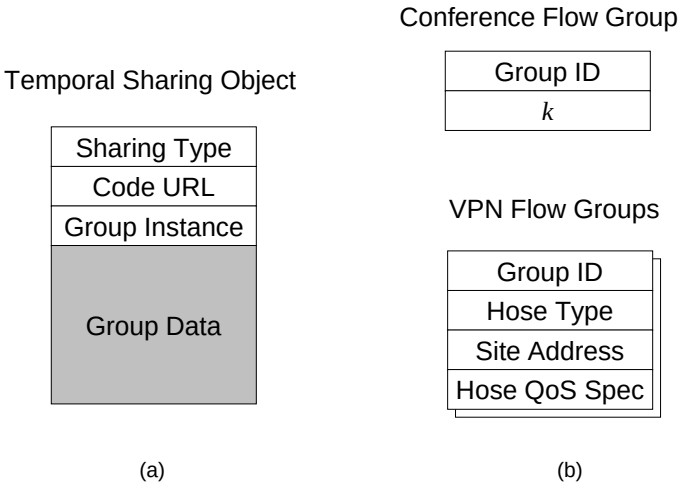

Fig. 6. (a) Temporal sharing object (b) Flow groups for conferencing and VPN sharing behaviors.

The overall design described above supports extensibility in two ways. The representation of temporal sharing is itself extensible by service providers as shown in the next section and the interpretation of temporal sharing information is by dynamically loaded code modules which can also be customized by service providers.

B. Temporal Sharing Representation

Temporal sharing information for each flow is represented by the `TemporalSharing` object shown in Figure 6(a). Each `TemporalSharing` object has a globally unique *sharing type* which represents a particular type of sharing behavior. Examples of sharing types are conferencing, VPN, etc. Associated with the sharing type is a *code URL* which provides the location of the code module that implements the temporal sharing behavior for that type. Beagle dynamically downloads the code module from the specified URL if it has not been downloaded before. Each `TemporalSharing` object also has a *group instance* field which uniquely represents an instance of a particular sharing behavior within an application. For example, if a VPN service creates two different VPNs, two different group instances of the VPN sharing type would be created. All flows at a link falling under the same group instance share the same set of resources at that link.

The `TemporalSharing` object also contains information which is opaque to Beagle (shown shaded in Figure 6(a)). This opaque data is interpreted by downloaded temporal sharing modules of that sharing type and contains information about one or more flow groups that the flow is a

member of. Figure 6(b) shows the information containe in flow groups for the conference and VPN sharing types Each flow group is identified by a *group id* which is uniqu within the application. The conference flow group also ha a parameter k which specifies the number of simultaneousl active sources. The VPN flow group has a *hose type* param eter which specifies whether the hose information pertain to a source hose or a destination hose. It also has parame ters to specify the address of the site connected to the hos and the QoS spec of that hose.

The group QoS spec for a flow group can also be speci fied in several different ways depending on the sharing styl Apart from the two ways of representing a flow group fo conference and VPN styles as shown above, a flow grou can use a general function $f(n)$ which calculates the re source requirement based on the n of flows actually shar ing the resource. An example of this would be statistica multiplexing of a number of video or audio flows.

Service providers can define custom representations for the `TemporalSharing` object by using service-specific flow groups. For example, a conference application that is also part of a VPN might define a flow group that combines the information in the conference and VPN flow groups described above. Broadcast-TV with PIP applications can define a flow group based on receiver preferences. Service providers defining custom representations for the `TemporalSharing` object must also provide the appropriate code modules to interpret the flow group data.

The design of the `TemporalSharing` object strongly couples a flow with a particular *sharing type*. This is because each flow setup message can carry at most one `TemporalSharing` object. Allowing multiple `TemporalSharing` objects in a flow setup message has the potential advantage of allowing applications to combine sharing styles (e.g. conference and VPN as described earlier). However, the disadvantage of this design is that it complicates the interaction between Beagle and the code modules that implement a particular sharing type. At each node, Beagle will need to interact with multiple sharing modules and will need to combine the results of these interactions in a generic way, which likely will make the computation of temporal sharing NP-complete. The design presented here does not preclude services from combining sharing styles. Instead, it forces services combining different sharing styles to implement a new sharing type and a sharing module for that type. This has the advantage of allowing services to use domain knowledge to simplify the calculation of the aggregate resource requirement.

C. Temporal Sharing Execution Environment Architecture

Figure 7 shows the design of the temporal sharing execution environment. The temporal sharing support primarily consists of three interacting modules: a) *core* Beagle, b) *temporal sharing manager* and c) *active sharing modules*. The core part of Beagle is responsible for flow setup protocol processing and maintaining flow state. This module does not interpret temporal sharing information and treats temporal sharing as an optimization. If temporal sharing information is not available or unusable on account of an error, this module falls back on independent flow-based allocation. As shown in Figure 7, the core part of Beagle maintains a list of group instances and the list of flows that fall under a particular group instance. Resource allocation is based on group instances with the traffic aggregate defined by the union of the filters of all flows that fall under that instance. The aggregate resource requirement for a group instance is provided by the sharing module corresponding to that particular sharing type. The core part of Beagle interfaces with the sharing modules through the temporal sharing manager.

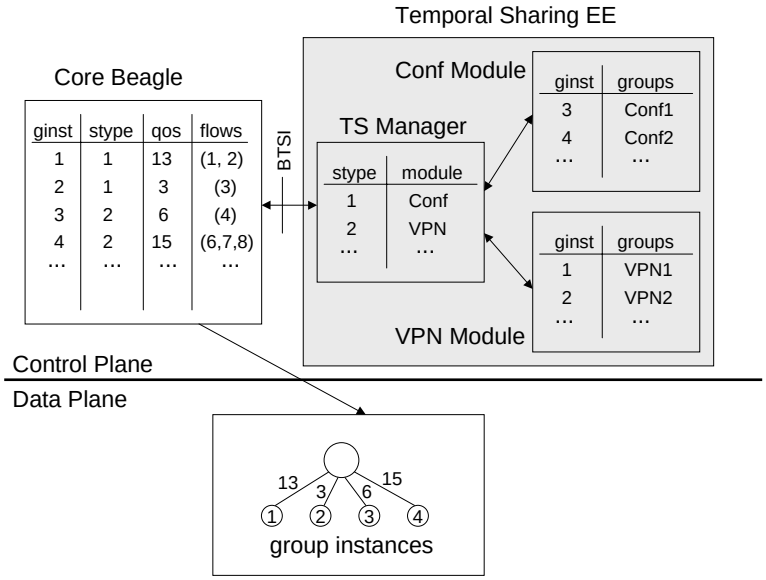

Fig. 7. Beagle extensible temporal sharing design.

The temporal sharing manager is responsible for the dynamic loading, instantiation and caching of sharing modules. It acts as an intermediary between core Beagle and the active sharing modules. The temporal sharing manager maintains a list of sharing types and references to corresponding sharing modules. When a new sharing type is received, the sharing manager obtains the code that implements the sharing module from the URL associated with the sharing type and instantiates the sharing module. The sharing manager is also responsible for directing requests from core Beagle to the appropriate sharing module. The sharing manager may also enforce computational and storage resource limits on the sharing modules. In addition, the sharing manager may act as a cache manager by maintaining all the sharing modules in a cache and paging out modules that have been inactive over a period of time.

Active sharing modules are responsible for implementing different sharing behaviors. Each sharing module is responsible for computing the aggregate resource requirement for a particular group instance. Each module can optionally keep flow group state for group instances of that type. For example, Figure 7 shows a VPN sharing module which has cached state for two VPN group instances. Keeping cached state allows a sharing module to optimize computational

TABLE II
BEAGLE TEMPORAL SHARING INTERFACE.

Interface call	Des
`TSI_Init`	Initialize group instance spe
`TSI_AddFlow`	Add flow to a particular gr
`TSI_DelFlow`	Delete a flow from a particu
`TSI_GetQoSSpec`	Get the aggregate QoS spec

overhead by using incremental calculation when setup or torn down. Each sharing module inter core Beagle through the sharing manager using t Temporal Sharing Interface (TSI).

The *Beagle Temporal Sharing Interface (TSI)* d interface between Beagle and the active sharing modules. Every sharing module must implement this interface. The TSI calls are shown in Table II. The `TSI_Init` initializes state in the sharing module for a particular group instance. This results in the downloading of the module if necessary. It returns an indication regarding whether the module has cached state for that particular group instance. If the module does not have cached state, Beagle is responsible for initializing the state for all the flows under that group instance. The `TSI_AddFlow` call adds a flow to a particular group instance. This call passes the opaque flow group data associated with that flow to the sharing module. The call returns an error indication if the flow group information is inconsistent or erroneous; or if the operation fails for some other reason. The `TSI_DelFlow` call deletes a flow from a particular group instance. The `TSI_GetQoSSpec` call returns the aggregate resource requirements for a particular group instance.

The TSI design presented here allows sharing modules to further optimize for computational overhead by performing incremental calculations using previously cached state for a group instance. For example, a conference sharing module may keep the flow QoS specs for each group instance in a sorted list and add the k highest QoS specs whenever the aggregate QoS spec for that group instance is requested. An alternative TSI design would be to pass all the flow QoS specs in a group instance to the sharing module every time the aggregate QoS spec needs to be calculated. This design has the advantage that the sharing modules can be stateless and therefore simpler to implement. However, the disadvantage is that sharing modules cannot amortize the cost of computation over several flow setups within a group instance. The TSI design presented in this paper also supports stateless sharing modules through the use of the `TSI_Init` call which indicates if a module has cached state. This provides the flexibility for applications to choose either stateless or stateful implementations for sharing modules.

The design of temporal sharing support in Beagle is driven by the goal of keeping the active sharing modules as simple as possible. Therefore, most of the functionality required to implement temporal sharing such as the definition of group instances, allocation of resources for group instances and arbitration of the shared resource during runtime are all implemented in the core non-extensible part of Beagle. The active sharing modules need only be concerned with calculating the aggregate resources for a particular group instance. This isolates the active sharing modules from the details of having to deal with the traffic control entities and simplifies the implementation of new sharing behaviors.

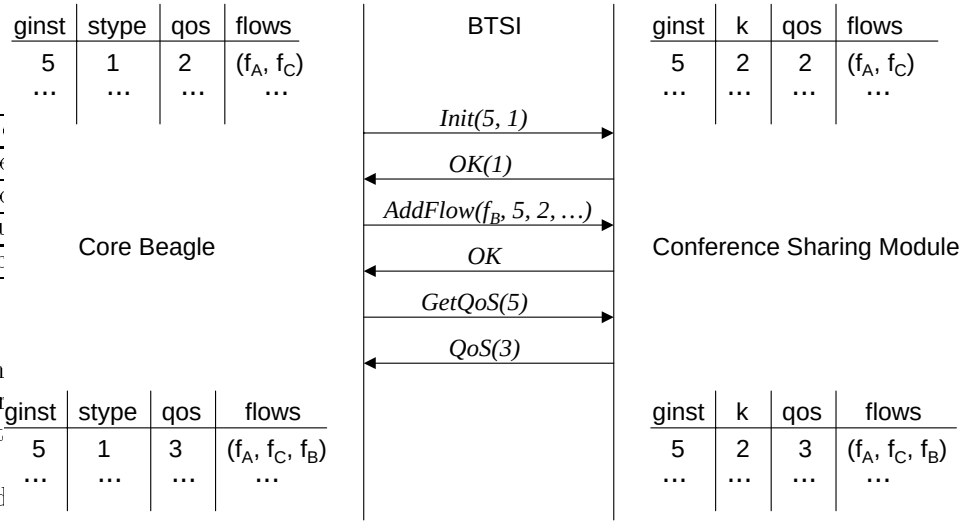

Fig. 8. Interaction between core Beagle and the conference sharing module during a flow setup.

Another design goal is to provide robust and predictable behavior in the presence of failures in the temporal sharing execution environment. Although extensibility provides applications with great flexibility in defining and implementing new sharing behaviors, the downside is the increased security risks and the possibility of errors in active sharing modules leading to unpredictable behavior. The design of the TSI guards against this possibility by providing a very simple interface that restricts the scope of actions that can be performed by the active sharing modules.

Another advantage of the design outlined here is that it allows the temporal sharing execution environment to operate without local knowledge about the network node which allocates the resources. This opens up the possibility of running the temporal sharing execution environment on a separate "control station" that is common for an entire subnet of routers in the network. This results in improved scalability and robustness; and can also enhance security.

D. Example

In this section we give an example to show the sequence of calls that are made across the TSI when a new flow is setup.

Consider the conferencing example shown in Figure 1. Figure 8 shows the interaction between Beagle and the conference sharing module at router R1 during the setup of the video flow from B (f_B) across the link between routers R1 and R2. We assume that the video flows from A (f_A)

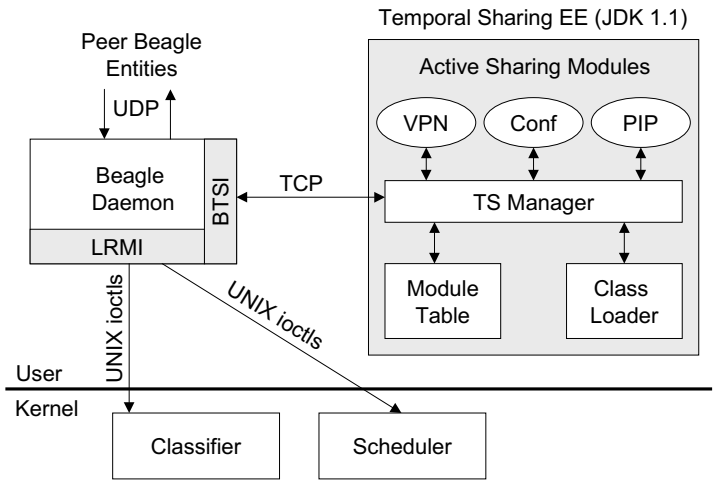

Fig. 9. Beagle temporal sharing implementation.

and C (f_C) have been setup previously. Therefore, before the setup of flow f_B, Beagle has allocated a bandwidth of 2 units at the link from R1 to R2. This is reflected in the state maintained by Beagle for the group instance that defines the video conference application as shown in the figure (group instance 5, sharing type 1). The conference sharing module also has cached state for this group instance as shown. When the flow setup message for flow f_B is received, Beagle first initializes the state for this group instance maintained by the sharing module by calling `TSI_Init` (shown as *Init(5,1)*). The sharing module indicates that it has cached state for this group instance in its reply (shown as *OK(1)*). Beagle then adds the new flow to the group instance by calling `TSI_AddFlow`. Upon receiving a positive response from the sharing module, Beagle obtains the new aggregate QoS spec for the group instance by calling `TSI_GetQoSSpec`. The sharing module adds flow f_B to the group instance and returns the newly computed aggregate bandwidth of 3 units in the response. This updates the resource allocation state maintained by Beagle for that group instance as shown in the figure. If the conference sharing module did not have cached state (i.e. it returns *OK(0)* in response to the `TSI_Init`), Beagle initializes the state by calling `TSI_AddFlow` for flows f_A and f_C before adding flow f_B.

VI. Implementation

In this section, we describe the implementation of the temporal sharing execution environment (TSEE) in the Beagle prototype. As shown in Figure 9, the TSEE interacts with the Beagle daemon using a TCP connection. We use the Java programming language to implement the active sharing modules based on its support for safe-execution of downloaded code modules, support for implementing security policies and wide-spread popularity. The TSEE is implemented as Java virtual machine process using JDK 1.1. The Beagle daemon is itself implemented in C and allocates resources for a flow using the traffic control interface (TCI) at a router.

TABLE III
Overhead of TSI calls in microseconds for conference style temporal sharing on a Pentium-II 400 MHz router running FreeBSD 3.3 with JDK 1.1.

TSI Call	Median	SIQR	10%	90%
`TSI_Init`	560.32	93.71	534.80	1153.01
`TSI_AddFlow`	465.50	10.26	454.16	737.95
`TSI_DelFlow`	471.83	10.07	457.68	951.95
`TSI_GetQoS`	443.52	9.15	437.20	1066.55

The main thread of control in the TSEE is the temporal sharing manager (TS manager). The TS manager maintains a module table that has references to downloaded sharing modules of a particular type. The module table can also be used to implement caching strategies. The TSEE also implements a class loader that can dynamically load classes that implement a particular sharing module given the code URL associated with that module.

The TSI is specified as a Java interface specification. Each active sharing module must define a class that implements this interface. Each sharing module can create multiple threads. The TS manager thread can control how much CPU is allocated to the sharing module by enforcing thread priorities. The TS manager acts as an intermediary between the Beagle daemon and the active sharing modules. It implements a serialization protocol across the TCP connection to the Beagle daemon that provides support for each TSI call. Each TSI call by the Beagle daemon causes the TS manager to invoke the corresponding method of the sharing module of that particular type. The values returned by the method invocation are serialized and passed back to the Beagle daemon.

The temporal sharing module of the Beagle daemon (shown shaded) implements the other end of the serialization protocol between the Beagle daemon and the TS manager. It provides an interface for the rest of the Beagle daemon to utilize services provided by the active sharing modules. Each TSI call is handled as a request-response transaction over the TCP connection. The temporal sharing module is also responsible for dealing with all the error conditions that might occur during any TSI transaction over the TCP connection.

VII. Evaluation

In this section, we present an evaluation of the Beagle temporal sharing design. In the data plane we demonstrate the operation of the VPN temporal sharing style by doing a proof-of-concept experiment using the Beagle prototype implementation over a local IP testbed. In the control plane, we profile the Beagle implementation and evaluate the cost of having an extensible implementation of temporal sharing.

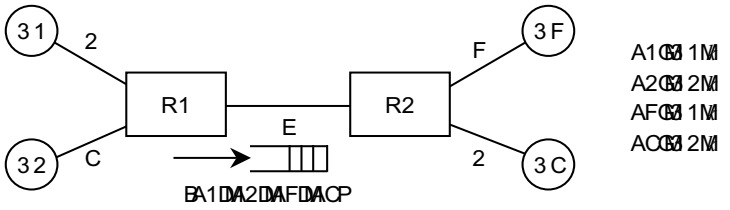

Fig. 10. Experimental setup for the hose-based VPN proof-of-concept experiment.

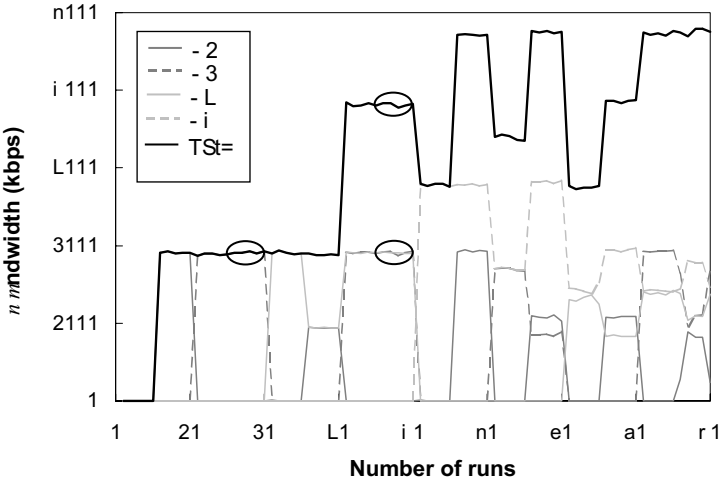

Fig. 11. Bandwidth plot of VPN flows along the link between R1 and R2.

A. Cost of Extensible Implementation

We measured the performance of the Beagle prototype implementation to determine the cost of providing extensible temporal sharing. The experiment involved repeated trials of setting up flows with conference type sharing on a local IP testbed. The overhead of invoking each of the TSI methods is shown in Table III. The setup of a flow with temporal sharing behavior involves at least three TSI calls (Init, AddFlow and GetQoS). There may more TSI calls if the sharing module does not keep cached state. Therefore the minimum cost of setting up a flow with temporal sharing is 1469.34 μs. This is in addition to the overhead of processing flow setup messages and allocating resources through the traffic control interface.

This overhead is caused mainly by two factors. Firstly, each TSI method call causes a context switch between the Beagle daemon and the Java process that runs the temporal sharing execution environment. This overhead is about 300 μs on each TSI call for a Pentium-II 400 MHz router running FreeBSD 3.3. We expect that commercial routers will offer better context switch performance by using real-time schedulers and/or multi-processor hardware. Another way to reduce this overhead could be to embed the temporal sharing execution environment within the Beagle daemon process using the Java Invocation API.

The second factor contributing to the overhead is the performance of sharing modules implemented in Java. For the experiment described here, we used JDK 1.1 which does not have support for just-in-time (JIT) compilation. We expect that the performance can be improved significantly with the use of JIT and advances in Java compiler technology.

It should be noted that both these factors contributing to the overhead are caused by implementation effects. We believe there is no fundamental reason why an extensible implementation should be slower than a non-extensible implementation.

B. Proof-of-concept VPN Experiment

In this section, we describe the results of a proof-of-concept experiment that shows the operation of the VPN application described in the example in Section V-D. Our goal is basically to demonstrate the operation of the Beagle prototype implementation of temporal sharing under a realistic experimental scenario. The experimental setup is shown in Figure 10. As shown in the figure, four hosts H1 through H4 represent four sites of the VPN. Each site has a hose bandwidth (in Mbps) as shown in the figure. For simplicity, we only consider the four flows crossing the link between routers R1 and R2. Using VPN style temporal sharing, the four flows share an aggregate bandwidth of 5 Mbps. Each of the four sources generates on-off TCP traffic. Each TCP flow can use as much bandwidth as is available subject to the constraints imposed by the hoses.

We cycle through each of the 16 on-off combinations of the four flows. For each combination, there are 5 runs of the experiment. During each run all the flows in the 'on' state transmit data for a period of several seconds. At the end of the run, throughput measurements are made for all the four flows at the receiving end. We plot the throughput for all the 80 runs of the experiment in Figure 11. Also plotted is the aggregate throughput of all the four flows (shown as the black solid line in the figure). The experiment demonstrates three aspects of temporal sharing. Firstly, as shown in the figure, the aggregate throughput for the four flows does not exceed 5 Mbps. This is due to the limitations imposed by the hoses on the aggregate traffic and shows that exploiting the limits imposed by the hoses leads to significant resource savings in the network. Secondly, each flow is capable of dynamically utilizing all of the available bandwidth within the limit determined by the minimum of its hoses at either end as shown in the figure. This demonstrates a second advantage of hose-based VPN sharing; networks can improve scalability by aggregating all of the VPN flows at a link under a single resource and at the same time provide sufficient flexibility for each end-to-end flow to dynamically utilize the maximum available bandwidth.

Finally, the experiment also shows the behavior of aggregate enforcement of temporal sharing. *Enforcement* of temporal sharing defines the behavior that results when the set of flows sharing a resource exceed the aggregate resource allocation. Aggregate enforcement simply enforces the limit on the aggregate traffic generated by a set of flows sharing a resource. In this case, packets that exceed the aggregate resource allocation are either dropped or forwarded

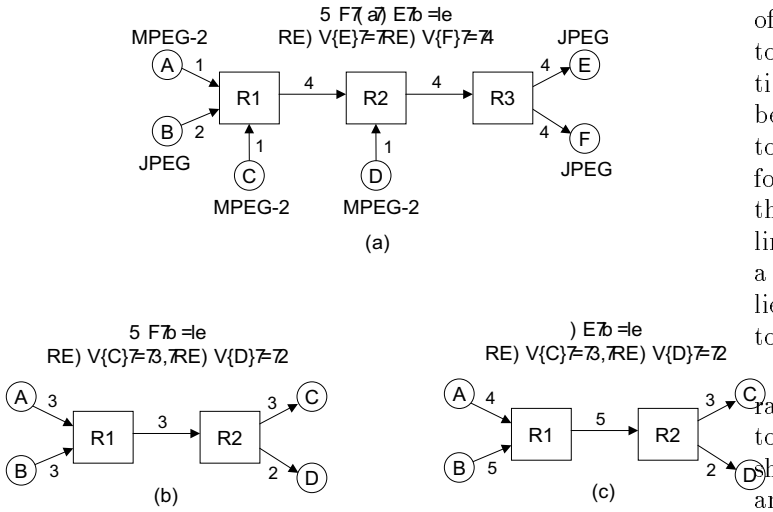

Fig. 12. RSVP temporal sharing examples: (a) Conference, (b) VPN with wildcard filter (WF) style, (c) VPN with shared-explicit (SE) style.

as best-effort. In most cases, as shown in Figure 11, aggregate enforcement provides fair behavior when two TCP flows share the aggregate bandwidth. However, in some cases, one TCP flow grabs all the bandwidth. These case are highlighted in Figure 11 using circles to show the runs where the unfair behavior occurs. For example, during runs 15-20, the flow F2 grabs all of the available 2 Mbps bandwidth of the destination hose of H4. This behavior can be remedied by using fair enforcement which ensures that both flows F1 and F2 would share this bandwidth equally using RED-like mechanisms [10], [11], [12]. Several other options for enforcement are described in [9].

VIII. Related Work

As mentioned before, temporal sharing has been studied in the context of other signaling protocols like RSVP [1], [2], Tenet-2 [3] and ST2+ [4]. In this section we contrast these protocols with Beagle showing how these protocols perform in the example scenarios considered in Section V-D.

Temporal sharing was first considered in the design of the RSVP protocol [1], [2], [13]. RSVP allows the sharing of reservations among different senders within a multicast session using reservation attributes called "styles". The RSVP version 1 specification defines three reservation styles. The wildcard filter (WF) style indicates that the resources reserved for a multicast session has to be shared by all senders to that session. The fixed filter (FF) style makes a separate reservation for each explicitly identified sender in the multicast session. The shared explicit (SE) style indicates that the resources reserved for the multicast session must be shared among the explicitly identified set of senders to the session.

The first difference between Beagle and RSVP temporal sharing is in the scope of its application. Temporal sharing in RSVP is restricted to sharing of resources within a multicast session. On the other hand, the Beagle notion of temporal sharing is more general in scope and applies to all the flows (multicast or unicast) within an application. The second difference is in the range of application behaviors supported. In RSVP, the FF style is analogous to independent flow-based resource allocation and therefore there is no sharing of resources in this case. Both the WF and SE reservation styles are designed with self-limiting applications in mind. In contrast, Beagle supports a much wider set of application behaviors as discussed earlier and provides an interface for applications and services to dynamically implement new sharing behaviors.

To see the difference between RSVP and Beagle temporal sharing we show the application of RSVP sharing styles to the examples of Section V-D in Figure 12. Figure 12(a) shows the resource allocations for the video conference example using either the WF or SE reservation style. Both styles produce the same resource allocations. Comparing with Figure 1(b), we see that RSVP over allocates resources on the inter router links. This is because the WF and SE styles associate a group QoS spec directly with a set of multicast flows. This leads to over allocation when all the flows in a conference do not have the same bandwidth requirement. When all the flows in a self-limited application have the same bandwidth requirement, both RSVP and Beagle temporal sharing styles produce the same allocations. Next, consider the resource allocation with RSVP for the VPN example shown in Figure 12. In this example we ignore the fact that RSVP cannot share resources between different unicast flows. We assume that the reservation styles can be extended in scope to cover multiple unicast flows. In this case, the WF style leads to under allocation of resources on the link between routers R1 and R2 (Figure 12(b)). This is because receivers C and D choose aggregate bandwidths corresponding to their hose limitations. When these reservations are merged upstream, the maximum of the two requests is allocated. On the other hand, using the SE style leads to over allocation along the access links of hosts A and B (Figure 12(c)). In this case we assume that both receivers C and D explicitly define the set of flows that share the aggregate resources defined by hose limitations. This set of flows is defined by all the flows converging at that particular receiver. With this assumption, resources are not merged upstream for these two sets of flows defined by C and D resulting in correct resource allocation on the link between R1 and R2. However, further upstream, independent flow-based allocation occurs because no two flows belonging to the same set share the same link. This example shows that RSVP reservation styles cannot adequately address the needs the network-limited applications which have multiple limitations on aggregate bandwidth for a set of flows.

Temporal resource sharing was also studied in the Tenet-2 signaling protocol. In the Tenet-2 model of temporal sharing, a list of *channels* can share resources if they belong to the same *channel group*. A channel group is analogous to the flow group defined in Beagle and defines an arbitrary association of flows for resource sharing purposes. Therefore, the Tenet-2 model has the same scope for the

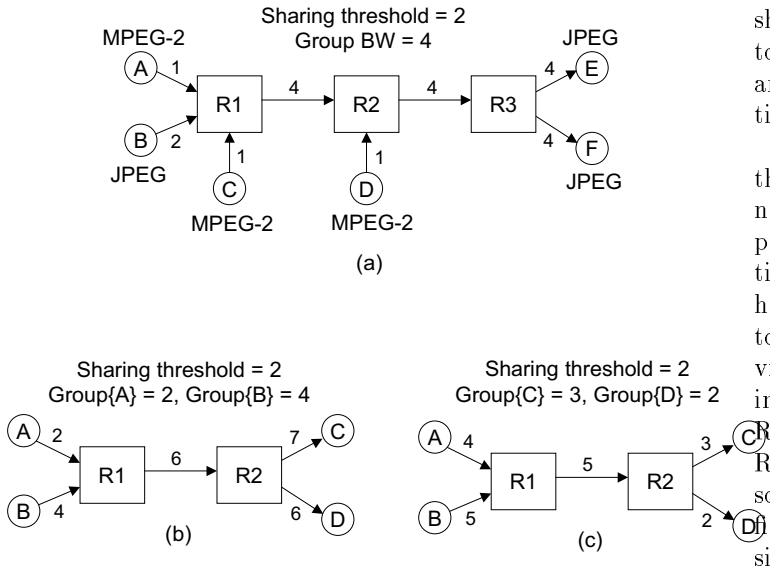

Fig. 13. Tenet-2 temporal sharing examples: (a) Conference, (b) VPN with source hose groups, (c) VPN with destination hose groups.

application of temporal sharing as Beagle. In the Tenet-2 model, the resource requirements for a channel group is given directly in terms of a group QoS spec. Associated with the group resource requirement is a *sharing threshold* that defines when the group requirement is to be used. When the number of channels at a link is greater than on equal to the sharing threshold, the group requirement is used at that link; otherwise, independent flow-based allocation is performed.

Figure 13 shows the resource allocations with the Tenet-2 temporal sharing model for the conference and VPN examples. As shown in Figure 13(a), the Tenet-2 model also results in over allocation of resources along the inter router links for the conferencing application. This is because, as with RSVP, Tenet-2 uses a group QoS spec to directly give the resource requirements for a group of flows. This is only accurate if all the sources in a conference have the same bandwidth requirement. In contrast, Beagle allows flow groups to have several different rules which give the aggregate resource requirement. The k-rule is appropriate for conferencing style applications and can handle sources with non-uniform bandwidth requirements as shown in Figure 1(b).

Another key difference between the Tenet-2 model and Beagle is that in Tenet-2 a channel can only be a member of one channel group with a resource sharing relationship. This means, like RSVP, the Tenet-2 model is mostly suited for self-limiting applications. Therefore, for network-limited applications like the hose-based VPN, applications can only satisfy one constraint on the aggregate bandwidth of a set of flows. This is shown by the application of Tenet-2 resource sharing to the VPN example where, either the source hose requirements are satisfied (Figure 13(b)) or the destination hose requirements are satisfied (Figure 13(c)). In either case, resources are over allocated when compared to the Beagle model of temporal sharing as shown in Figure 2(c). The Internet stream protocol ST2+ also provides support for sharing of bandwidth among multiple streams. The sharing model is almost identical to that provided by the Tenet-2 scheme.

The Beagle design for temporal sharing uses ideas from the active signaling project [14], [15] at ISI. The active signaling project is developing an active version of the RSVP protocol that can be dynamically customized by applications and service providers. The active signaling project has mainly been concerned with defining the proper protocol programming interfaces and designing execution environments using which active protocols can be designed, implemented and tested. Currently, an active version of the RSVP protocol has been developed where, a base version of RSVP co-exists with an enhanced version which supports some of the optional features of the RSVP protocol specification. The Beagle design incorporates ideas from active signaling into the design of the temporal sharing execution environment.

IX. Conclusions

This paper made the case that signaling support of temporal sharing must be extensible by applications. To support this argument we first presented several classes of applications that exhibit different styles of temporal sharing. Then the notion of flow types was introduced which enables the characterization of the temporal sharing design space using a two-dimensional classification defined by the two types of flows: related and independent. Using set theory, we showed that supporting the most general forms of temporal sharing for the two types of flows is computationally intensive. We also showed how several useful styles of temporal sharing can be supported cheaply either by using application domain knowledge, or by trading off resource efficiency for computation overhead.

This paper also presented the design of extensible temporal sharing support in the Beagle signaling protocol. Experimental evaluation of the Beagle prototype implementation shows that the overhead incurred by implementing the extensible parts in Java is reasonable (about 500 μs per call) and will improve with better support for real-time scheduling on commercial routers and improvements in Java compiler technology.

We also presented results of a proof-of-concept experiment to demonstrate the use of the hose-based VPN style to save resources for a VPN service. The Beagle temporal sharing design was contrasted with other signaling protocols using several examples which clearly showed the benefits of extensible support for temporal sharing.

References

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

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

[3] Amit Gupta, Winnie Howe, Mark Moran, and Quyen Nguyen. Resource Sharing in Multi-Party Realtime Communication. In *Proceedings of INFOCOM 95*, pages 1230–1237, Boston, MA, April 1995.

[4] L. Delgrossi and L. Berger. Internet Stream Protocol Version 2 Protocol Specification - Version ST2+, August 1995. Internet RFC 1819.

[5] D.J. Mitzel and S. Shenker. Asymptotic Resource Consumption in Multicast Reservation Styles. In *Proceedings of the SIGCOMM '94 Symposium on Communications Architectures and Protocols*, pages 226–233, August 1994.

[6] N.G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K.K. Ramakrishnan, and J.E. van der Merwe. A Flexible Model for Resource Management in Virtual Private Networks. In *Proceedings of ACM SIGCOMM '99 conference on applications, technologies, architectures, and protocols for computer communication*, pages 95–108, Cambridge, September 1999.

[7] Prashant Chandra, Allan Fisher, and Peter Steenkiste. A Signaling Protocol for Structured Resource Allocation. In *IEEE INFOCOM'99*, pages 522–533, New York, March 1999.

[8] Steven Blake, David Black, Mark Carlson, Elwyn Davies, Zheng Wang, and Walter Weiss. An Architecture for Differentiated Services, December 1998. IETF RFC 2475.

[9] Dr. Anonymous. *A Signaling Protocol for Value-added Network Services*. PhD thesis, Department of XXX, YYY University, 2000.

[10] Sally Floyd and Van Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[11] Dong Lin and Robert Morris. Dynamics of Random Early Detection. In *Proceedings of the SIGCOMM '97 Symposium on Communications Architectures and Protocols*, pages 127–137, Cannes, August 1997.

[12] Ion Stoica, Scott Shenker, and Hui Zhang. Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks. In *Proceedings of the ACM SIGCOMM '98 conference on applications, technologies, architectures, and protocols for computer communication*, pages 118–130, Vancouver, September 1998.

[13] R. Braden, D. Estrin, S. Berson, S. Herzog, and D. Zappala. The Design of the RSVP Protocol, Final Report, June 1995. Available from the URL http://www.isi.edu/rsvp/DOCUMENTS/rsvp1.final.report.ps.

[14] B. Braden, A. Cerpa, T. Faber, B. Lindell, G. Phillips, and J. Kann. ASP EE: An Active Execution Environment for Network Control Protocols, May 1999. Available at URL http://www.isi.edu/active-signal/ARP/DOCUMENTS/ASP_EE.ps.

[15] G. Phillips, B. Braden, J. Kann, and B. Lindell. ASP PPI: An Active Execution Environment's Protocol Programming Interface, May 1999. Available at URL http://www.isi.edu/active-signal/ARP/DOCUMENTS/PPI.ps.