Dynamic Resource Migration for Multiparty Real-Time Communication

Riccardo Bettati[*] **and Amit Gupta**
{bettati,amit}@icsi.berkeley.edu

The Tenet Group
University of California at Berkeley, and
International Computer Science Institute

TR-95-060
October 1995

Abstract

With long-lived multi-party connections, resource allocation subsystems in distributed real-time systems or communication networks must be aware of dynamically changing network load in order to reduce call-blocking probabilities. We describe a distributed mechanism to dynamically reallocate ("migrate") resources without adversely affecting the performance that established connections receive. In addition to allowing systems to dynamically adapt to load, this mechanism allows for distributed relaxation of resources (i.e. the adjustment of overallocation of resources due to conservative assumptions at connection establishment time) for multicast connections. We describe how dynamic resource migration is incorporated in the Tenet Scheme 2 protocols for multiparty real-time communication.

[*]Now at: Department of Computer Science, Texas A&M University, 301 H.R. Bright Building, College Station, TX 77843-3112. Tel: (409) 845-5469. Fax: (409) 847-8578

1 Introduction

The increasing speed of computer networks and the improvement of workstation capabilities are enabling a new class of distributed applications, those involving multimedia data. It is widely believed that these applications should be supported in the general framework of real-time communication [1, 2, 3, 4, 5, 6], which provides predictable performance (e.g., end-to-end delay bounds for data delivery); typically, the clients negotiate with the network service provider to obtain a desired Quality of Service (QoS), which the provider guarantees. A number of schemes and protocols have been proposed to provide real-time communication services. Most of the schemes are connection-oriented, and reserve resources (bandwidth, buffers, and so on) along the route of a real-time connection [1, 2, 7]. QoS guarantees cannot be provided if the network does not check for saturation before accepting new channels. Thus, the network must perform *admission control* to ensure that the guarantees are met.

This resource reservation is an on-line process; whenever the establishment of a new channel is requested, the routing system computes the "best" route (according to some optimality criteria that may be based on current workload, residual capacity, future traffic characterization and blocking probability computation). The signalling protocol then contacts the local resource managers (LRMs) at each server[1] along the channel route to allocate resources. For services that provide bounds on end-to-end delay, at least three distinct approaches are possible:

- The LRM allocates resources based on the current workload. The amount of resources allocated does not depend on the end-to-end performance parameters that the client requires. This is the approach that RSVP takes [9].

- The LRM provides, to the signalling protocol, a range of feasible resource allocations; in a separate pass, the signalling protocol chooses the final resource allocation. This is the approach followed in OPWA [11].

- In a first pass, the LRM makes a tentative resource allocation. The signalling protocol then determines the end-to-end performance provided at the current allocation levels; if this exceeds the performance bounds requested by the clients, the signalling protocol can, in the second pass (called *relaxation pass*), inform the LRM to reduce (*relax*) the resource allocations [8, 10] appropriately. This is the approach that the Tenet protocols have taken [8].

With the first approach, the resource allocation decisions for a channel are oblivious of the channel's end-to-end performance bounds. This may lead to inefficiencies in resource allocations, as well as the network not meeting the end-to-end performance bounds. A "smart" resource allocation mechanism that understands the

[1]A server here refers to a node or a link.

end-to-end performance bounds can, in principle, make better resource allocation decisions, thereby meeting the performance bounds for more channels.

With the second approach, the key problem is in managing the information and in limiting the combinatorial explosion of choices and alternatives. One approach [11] is to insist that the information be provided as a function of a set of parameters (e.g., that the local delay bound will be of the form $a + \frac{b}{x}$, where a and b are provided by the LRM, which also provides bounds for values of x). The signalling protocol then accumulates this information to obtain the feasible set for end-to-end performance parameters, decides the performance bounds that the channel will obtain (this decision may be based on further interactions with the clients) and inform the LRMs accordingly.

With both the second and the third approaches, a critical problem is that the network load changes dynamically; hence it is desirable that the network be able to dynamically adapt the resource allocation decisions to the dynamically varying workload; this can lead to significant reduction in call blocking probabilities for connection acceptance. It can also make simpler initial resource allocation mechanisms much more attractive; these simpler mechanisms in turn make channel set-up fast and scalable.

Of course, such dynamic changes should not adversely affect the performance that the active channels receive; it is desirable that distortions, if any, be small. In any case such distortions should be within the performance bounds specified by the client.

In this paper, we will describe a set of mechanisms (referred to as Dynamic Resource Migration, or DRM) that dynamically change resource allocations (thereby "migrating" resources from node-to-node in the network) in a manner in which resources are freed around the congested parts of the network and such migration does not adversely impact the performance guarantees made to the service clients.

This paper is organized in the following manner. Section 2 describes Tenet Approach real-time communication, in particular multi-party real-time communication. We discuss the basic mechanisms for dynamic resource reallocation in Section 3. In Section 4, we discuss how these mechanisms can be incorporated into distributed schemes for appropriate load balancing in real-time networks. We conclude this discussion in Section 5 with a summary and an outlook on future work.

2 Background

In this paper, we illustrate our approach to dynamic resource migration (DRM) in the framework of the Tenet multi-party real-time communication protocols [2]; though this discussion is limited to the Tenet protocols, the underlying ideas and techniques are equally applicable to other multi-party real-time communication protocols. Our previous paper [10] describes the relevant aspects of Tenet; in this discussion, we limit ourselves to discussing a few of its salient features. We first describe the Tenet approach to real-time communication and follow the discussion with a description of key issues that arise in multi-party settings. We will then describe two aspects of the Tenet Suite 2: how state information is currently maintained, and how the channels are set-up so as to meet the client-specified performance requirements. This discussion thus sets the stage for describing our proposed mechanisms for dynamically migrating resources in the Tenet Suite 2.

2.1 The Tenet Approach to Real-Time Communication

The scheme on which the Tenet Suite 2 is based implements the *multicast real-time channel* abstraction. This communication abstraction is defined as a simplex connection between a source and a set of destinations, capable of guaranteeing a given (and possibly different) quality of service (QoS) to each destination. A real-time channel ("channel" for short) is characterized in the Tenet schemes at the network layer by a set of traffic specification parameters and, for each destination, a quadruple of QoS parameters that specify the desired end-to-end performance. Scheme 2 allows channels to be established from the source or from the destinations; we describe here, for brevity, only the former procedure.

When a client wants to set up a channel, it invokes the Real-Time Channel Administration Protocol (RCAP) and passes to it suitable identifiers for the source and the destinations, as well as the source's traffic parameters and each destination's QoS parameters. RCAP gets a possible route for the channel from a Routing Server, and issues a *channel-establish* message from the source. This message follows the given route, replicating itself at each branch node it encounters on its path, and sending a copy of the message down each of the branches of that subtree. Admission tests are performed at each server that is reached by a copy of the message; if any of the tests is unsuccessful, a *channel-reject* message is return to the source; if all tests are successful, a copy of the message is sent to the next server on that path, until it reaches one of the destinations. Each destination makes a final decision about the establish request for the new channel, and returns an *channel-accept* or *channel-reject* message to the source. Messages of both types wait for those from the destinations in the same subtree at the subtree's branch node, where they are merged before the resulting message is forwarded on the reverse path toward the source.

Some resources had been tentatively reserved by the *channel-establish* message or one of its replica in each traversed server. When receiving the returning message(s), which will generally contain both accepts and rejects from the various destinations in the corresponding subtree, the server cancels those reservations (if all the replies are of the *channel-reject* type) or adjusts them according to the information received from each accepting destination. The process of adjusting the resource requirements and reducing them to satisfy the needs of accepting destinations is called *resource relaxation.* These adjustments are reflected in the return message forwarded to the server immediately upstream, as well as in the amounts of resources actually reserved in the server for the new channel. At the end of this process, the source receives the results of the request in a single return message, and transmits them to the client for evaluation and further action.

The newly created channel reaches all destinations that have accepted the establishment request and is now ready for immediate use. It remains in existence until it is torn down by the client.

2.2 Multi-Party Issues in Real-Time Communication

Many multi-party applications involve a large number of recipients for each data stream; it is clear that multicasting can be used to reduce the traffic on the network nodes and links thereby saving valuable network resources.

A key component of the multi-party communication is the presence of multiple senders and receivers. A strawman multicast scheme would require that at the connection establishment time, the sources specify the list of receivers. It is unreasonable to require that in a large-scale distributed multimedia application (e.g. computer-based-conferencing) the sender (or for that matter, any central application-based authority) knows about all the receivers; it is equally unreasonable to require the receivers to know about all potential senders for that conference. It is important that the network service supports this decoupling between the different participants; the network should provide the rendezvous among the participants interested in a common "session".

The realtime nature of the conference also favors this separation of the senders and receivers. It is expected that different receivers will be heterogeneous, i.e. that they will vary both in their ability to handle the data, and in the QoS requirements they may have. It is generally unreasonable to expect the senders to specify these properties for all possible destinations of their data stream; this will also not scale well to larger conferences.

Multi-party conferences tend to be long-lived, i.e., participants may join (or leave) a session while it is in progress. It is important that the network service provide support for dynamic changes in the group of participants.

For supporting the abovementioned aspects of multi-party communication, the key abstraction is the *real-time multicast group*, also referred to as the *target set* abstraction. This target set abstraction is the real-time analog of the IP hostgroup abstraction, in that while an IP hostgroup has, as members, the destinations interested in listening to a common "session", the target set members are these interested destinations along with the requested bounds on end-to-end performance (e.g., end-to-end delay, the jitter, i.e., variation in the delay etc.). A channel logically transmit data from a particular sender to a target set; this amounts to transmitting the data from that sender to all members of the target set. Receivers can dynamically join and leave a target set; when they join a target set, they start getting data on all channels sending data to the target set. In this manner, the target sets support the decoupling between the senders and the receivers and also provide the rendezvous among them.

Traditional real-time network systems (e.g., [2]) treat traffic on different connections independently when determining their resource requirements. For multi-party real-time communication, this results in inefficient over-allocation of resources [12]. For example, consider an audio-conference of one hundred persons. In the strawman proposal, the conference is set up by establishing one hundred multicast channels, one from each speaker (sender) to all listeners (destinations). It is reasonable to expect that only one person speaks at any time. Along common sub-paths (for these hundred channels) it would be sufficient to reserve resources for two audio channels (to allow some over-speaking). Thus, the resource allocation can be reduced (and the allocation efficiency increases) if the network clients can specify these *resource sharing* properties to the network and if the network can use such information to reduce the resource allocation along common sub-paths.

The Tenet Scheme 2 provides channels groups [13]; the *resource sharing* channel groups allow the network clients to specify these resource sharing relationships to the network. In the above example, the application would: (a) create a new channel group, and (b) inform the network to include the hundred audio channels in this channel group. The client would also inform the network that at any server in the network, the aggregate resource allocation for all channels should not exceed two audio channels. During channel establishment for these channels, at any server, the admission test system can determine if it has already allocated resources for two audio channels and if so, accept this new channel without allocating any more resources. This mechanism fits in especially well with the rest of the Tenet scheme because it is fully-distributed; different servers make this decision independently.

2.3 Resource Allocation for Multiparty Real-Time Communication

The two-pass resource allocation procedure for channel establishment described in Section 2.1 has proven effective for two-party communication settings with unicast channels. Its commitment of resources at establishment time makes it inappropriate, though, for longer-lasting multiparty sessions, which experience a dynamically changing environment during their lifetime: on one side, the load of the network changes; on the other, the number of participants, or even their geographic distribution, dynamically varies. A more appropriate resource allocation scheme would allow to reconsider the commitment of resources made at channel establishment time to adapt to both changes in the load of the underlying network and in the participant population connected to the channel.

Typically, the decision of how much of a particular resource to allocate to the new channel at a node is not a local one. In general, resource requirements are interdependent across nodes. The additive behavior of local delays is the simplest example of such an interdependence: a decrease in priority allocation at one node (causing an increase of the local delay) has to be compensated by an increase of priority at another node (which decreases the local delay at that node) to maintain the overall end-to-end delay. Generally, interdependencies are more subtle. For instance, the priority assigned to a channel at a node defines the amount of jitter introduced to packets of that channel by that particular node. Hence, the amount of priority resource allocated to the channel at the node controls the amount of buffer needed by the next nodes downstream to make up for the introduced jitter. (In the following discussion, we will use this simple relation between allocated priority resource on a node and the required buffer resource on the nodes directly downstream to illustrate the interdependence of resource requirements across nodes.)

When it comes to relaxing the tentative resource allocations, a global relaxation scheme must be used in order to account for these interdependencies in resource requirements. In the case of unicast channels, the relaxation can be done locally, based on a combination of (1) information contained in the *establishment-accept* message flowing back to the source, and (2) an implicit global relaxation policy.

Such a scheme can not be used for multicast real-time channels because the interdependence between a branch node and its descendants effectively introduces interdependence relations between resource allocations across the subtrees. More specifically, the amount of resources that can be relaxed in the nodes in one subtree depends on the resource requirements in another subtree, and vice versa. This can not be resolved in the two-pass scheme described earlier, but would require an exponential number of passes. The cost of channel establishment for any non-trivial multi-party real-time channel would be non-acceptable.

With the proper resource reallocation procedures in place, it is possible to apply a very conservative relaxation scheme (for instance, no relaxation at all) during the establishment. Once the channel is established, resource reallocation can then be used to dynamically relax the resource requirements.

3 Mechanism

In a dynamic resource reallocation scheme, nodes must be able to renegotiate resource allocations at any point during the lifetime of an established channel. For example, after a channel has been established, a number of nodes on it can become heavily loaded due to other, newly established channels, and require resources to be reallocated. In other cases, the participants (and their performance requirements) of the session using the channel may change enough to warrant a reconsideration of the resources allocated to the channel at establishment time. In order to be efficient, and to reduce the granularity of change to which such a scheme can adapt, the renegotiations must be light-weight. Thus, the number of nodes involved in a renegotiation must be kept small. An example of the opposite approach is Dynamic Channel Management (DCM) [14], where the renegotiation consists of a re-establishment of the entire channel. Although particular attention is given in this scheme to minimizing transient over-allocation of resources during the re-establishment process, DCM is not easily scalable, in particular for multicast channels. In designing a dynamic resource allocation scheme, it is important to ensure that the QoS guarantees provided by active channels (including the channel being reallocated the resources for) are not affected by the resource reallocation process, neither during nor after the reallocation. In this section, we describe a local resource reallocation mechanism, which allows to dynamically reallocate resources while involving only a small number of nodes; in fact, three adjacent nodes participate for unicast portions of the channel (we call them *primary, secondary*, and *tertiary* node, N_p, N_s, and N_t, respectively), and $O(N)$ nodes participate when primary or secondary nodes with out-degree N are involved. In addition, dynamic resource reallocation does not affect the QoS guarantees provided to the receivers.

In the following, we describe the steps involved in resource renegotiation. For sake of simplicity of argument, we will assume a rate-controlled fixed-priority (RCSP) scheduling discipline [15] on all the nodes. With the appropriate modifications, the description below is valid for other scheduling disciplines as well, such as jitter-controlled earliest-deadline-first (J-EDD) [16], stop-and-go [17], hierarchical round robin [18], and combinations thereof.

Three types of resource reallocation can be distinguished, depending on the location in the multicast tree:

1. Resource reallocation in a unicast portion of the multicast: None of the participating nodes is a branch node or a receiver node.

2. Resource reallocation involving branch nodes: At least one participating node forwards data to more than one successor in the multicast tree.

3. Resource reallocation involving a receiver node: At least one participating node serves a receiver.

Before describing the reallocation steps in more detail, we define some terms: Let the nodes along a particular path of an established channel C be labeled $(N_1, N_2, \ldots, N_m)$. Before the reallocation, each node N_i has allocated buffer space B_i, a scheduling priority ϕ_i to channel C, resulting in a worst-case delay d_i for packets on C. The value of d_i depends on the scheduling algorithm used, and is typically determined by time-demand analysis. The amount of buffer space that must be allocated for C at node N_i is defined by [15]:

$$B_i = \left(\left\lceil \frac{d_{i-1}}{X_{min}} \right\rceil + \left\lceil \frac{d_i}{X_{min}} \right\rceil \right) P \quad ,$$

where X_{min} and P are the worst-case packet inter-arrival time and worst-case packet length, respectively.

In the following, we describe the reallocation process in detail. We first describe the unicast case to present the general idea of the procedure. In Section 3.2 we show how the basic procedure is extended to branching portions of the multicast tree. In Section 3.3 we illustrate how the same procedure is used to effectively "trade in"

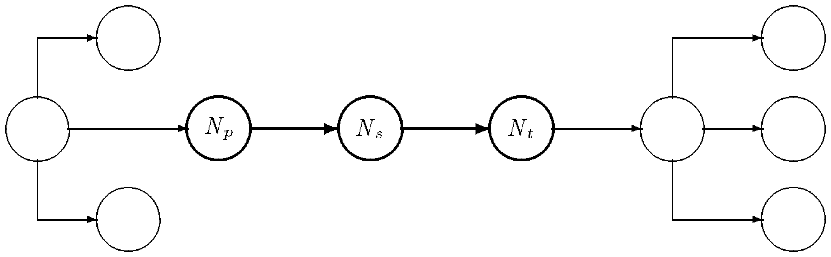


Figure 1: Unicast Portion of a Multicast Tree.

excessive QoS (typically slack) at the receiver for allocated resources, thus allowing the realization of resource relaxation through dynamic reallocation.

In the following descriptions we ommit formal proofs that the reallocation process does not violate QoS guarantees of active channels. We note however, that it is easy to verify that at no time the end-to-end delay or the jitter are increased during the reallocation.

3.1 Resource Reallocation in a Unicast Portion of the Multicast Tree

The simplest case for resource reallocation is the case when none of the participating nodes is a branch node of the multicast tree. Figure 1 illustrates this case: The reallocation procedure consists of four steps, which we describe in the following for the case of reducing the load on N_p by reducing the priority ϕ_p and increasing ϕ_s:

Step 1: *Increase priority on N_s:* Set $\tilde{\phi}_s$ to be the lowest priority that guarantees a local delay $\tilde{d}_s \leq d_s + d_p - \tilde{d}_p$, where $\tilde{d}_p$ denotes the local delay bound on N_p after the priority reduction.

Step 2: *Increase buffer size on N_s:* Set

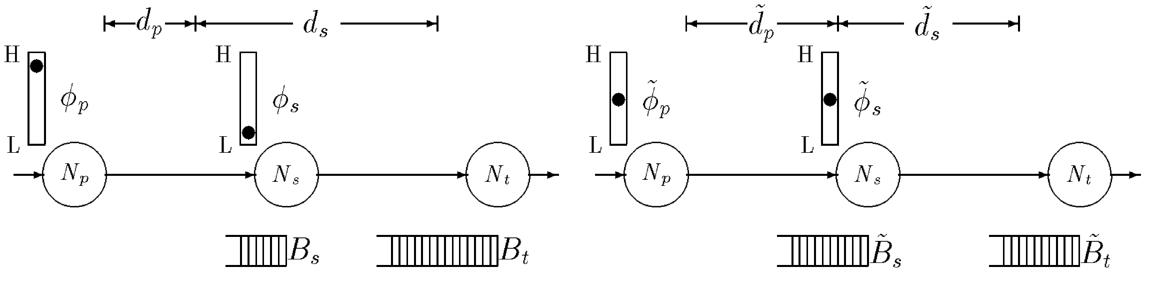
$$\tilde{B}_s := \left(\left\lceil \frac{\tilde{d}_p}{X_{min}} \right\rceil + \left\lceil \frac{\tilde{d}_s}{X_{min}} \right\rceil \right) P \quad .$$

Step 3: *Decrease priority on N_p:* Set the priority on N_p to be $\tilde{\phi}_p$.

Step 4: *Decrease buffer size on N_t:* Set

$$\tilde{B}_t := \left(\left\lceil \frac{\tilde{d}_s}{X_{min}} \right\rceil + \left\lceil \frac{d_t}{X_{min}} \right\rceil \right) P \quad .$$

Figure 2 illustrates this form of resource reallocation: Before the reallocation, packets of channel C are served at high priority on node N_p and on low priority on



(a) Before Reallocation. **(b)** After Reallocation.

Figure 2: Resource Reallocation on Unicast Portion of Multicast Tree.

node N_s. This reflects on the local worst-case delays and on the buffer sizes: both the the local worst-case delay and the buffer requirement on N_s are smaller than on N_t. After the reallocation, the scheduling priority of C on N_p is reduced, causing the worst-case delay to increase. This is compensated by the increased priority on N_s. While increasing the priority on N_s reduces the buffer requirements on N_t, the reduced priority on N_p causes the buffer requirements on N_s to increase. Overall, this example illustrates how resource reallocation can balance the resource requirements.

To conclude, we note that on N_t the resource reallocation in this case affects the buffer requirements only. Therefore, this reallocation scheme also works when N_t is a branch node in the multicast tree.

3.2 Resource Reallocation Involving Branch Nodes

When a branch node participates in the reallocation, this affects all its successors as well. This is illustrated in Figure 3, where the branch node N_p is the primary node of a resource reallocation. We note that in this example some of the secondary nodes ($N_s^{(1)}$ and $N_s^{(2)}$) are branch nodes too. In the following, we define $\mathcal{N}_s$ to be the set of all secondary nodes $N_s^{(1)}, N_s^{(2)}, \ldots$, and $\mathcal{N}_t$ to be the set of all tertiary nodes $N_t^{(1)}, N_t^{(2)}, \ldots$. In other words, $\mathcal{N}_s$ denotes the set of all successors of the primary node, and $\mathcal{N}_t$ the set of all successors of nodes in $\mathcal{N}_s$. The resource reallocation procedure, for the case of a reduction of the priority ϕ_p on node N_p, looks as follows:

Step 1: *Increase the priority on nodes in $\mathcal{N}_s$:* On all $N_s^{(i)}$, set $\tilde{\phi}_s^{(i)}$ to be the lowest priority that guarantees a local delay $\tilde{d}_s^{(i)} \leq d_s^{(i)} + d_p - \tilde{d}_p$, where $\tilde{d}_p$ denotes the local delay bound on N_p after the priority reduction.

Step 2: *Increase the buffer size on nodes in $\mathcal{N}_s$:* On all $N_s^{(i)}$ in $\mathcal{N}_s$, set

$$\tilde{B}_s^{(i)} := \left(\left\lceil \frac{\tilde{d}_p}{X_{min}} \right\rceil + \left\lceil \frac{d_s^{(i)}}{X_{min}} \right\rceil \right) P \quad .$$

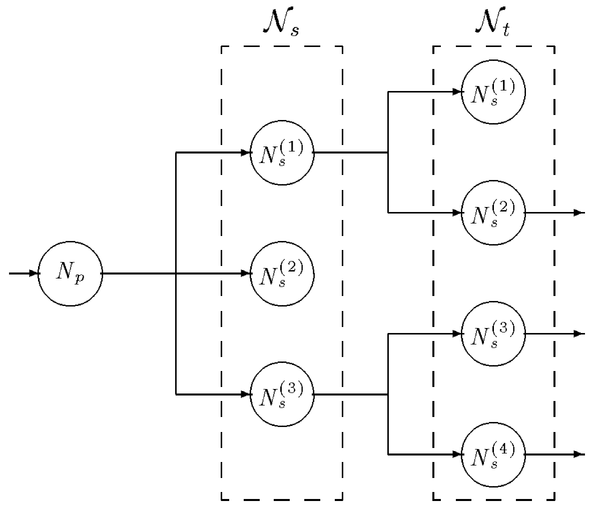


Figure 3: A Branching Portion of the Multicast Tree.

Step 3: *Decrease priority on N_p:* Set the priority on N_p to be $\tilde{\phi}_p$.

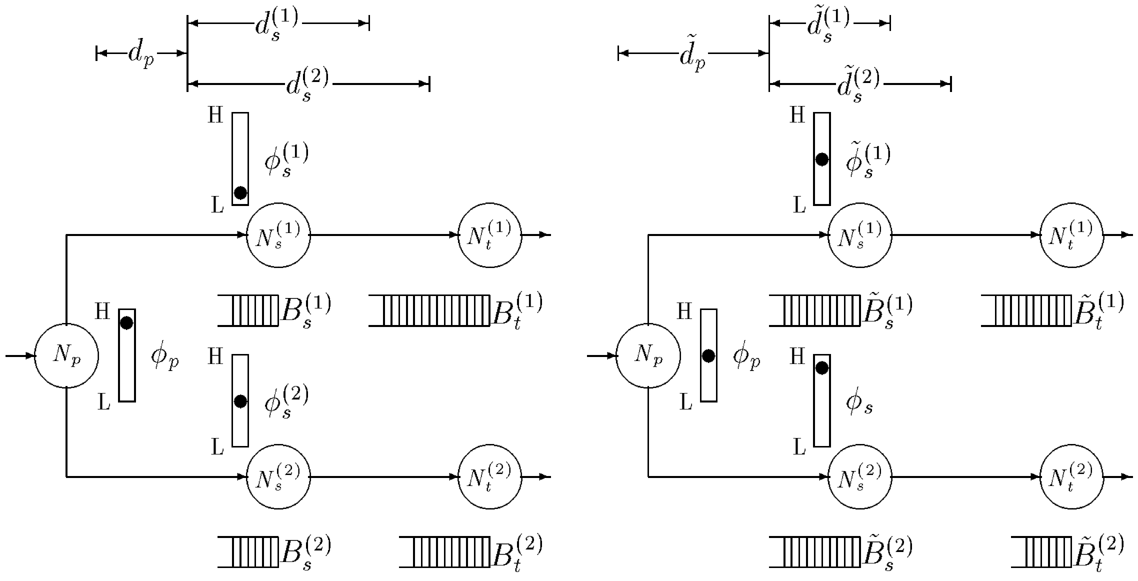
Step 4: *Decrease buffer sizes on nodes in $\mathcal{N}_t$:* On all $N_t^{(i)}$ in $\mathcal{N}_t$, set

$$\tilde{B}_t^{(i)} := \left(\left\lceil \frac{\tilde{d}_s}{X_{min}} \right\rceil + \left\lceil \frac{d_t^{(i)}}{X_{min}} \right\rceil \right) P \quad .$$

Figure 4 illustrates this procedure for the example of a primary branch node with two successors, $N_s^{(1)}$ and $N_s^{(2)}$: Before the reallocation, the priorities assigned to the two secondary nodes are medium to low, whereas the primary node has assigned a high priority to the channel. The reduction of priority on the primary node is compensated by an appropriate increase of priority and buffer requirements on both secondary nodes.

3.3 Resource Reallocation Involving a Receiver Node

Receiver nodes (i.e. nodes that host a receiver of the channel) play a special role in the resource reallocation scheme. We remember that during the final acceptance test at establishment time, the receiver makes the final decision about the acceptance of a channel by testing the performance it provides. For instance, the receiver node tests (1) the delay by comparing the end-to-end delay to the sum of local delays along the path from the source (the difference, if positive, is called the *slack* s_i of node N_i), and (2) the jitter of the channel to be established. As we will see, slack can later be "traded in" while the channel is active, during future resource reallocations. Resource reallocation involving receiver nodes comes in two forms, depending on whether the receiver is on the primary or the secondary node. We describe the simpler reallocation procedure for the case of the receiver node being the primary node: In this case, the



(a) Before Reallocation. (b) After Reallocation.

Figure 4: Resource Reallocation Involving a Branch Node.

slack in the receiver node can be used to trade for a reduction of the priority at the primary node. The reallocation procedure in the case of a reduction of priority ϕ_p looks as follows:

Step 1: *Decrease slack on N_p:* Set $\tilde{s}_p := s_p + d_p - \tilde{d}_p$, where $\tilde{d}_p$ denotes the local delay bound on N_p after the priority reduction.

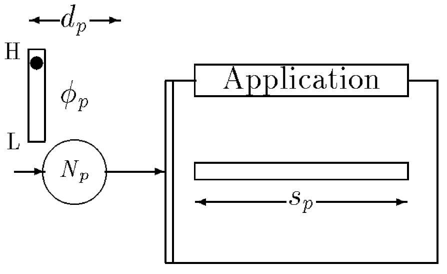
Step 2: *Decrease priority on N_p:* Set the priority on N_p to be $\tilde{\phi}_p$.

We notice that during this reallocation procedure, no buffers are reallocated.
Figure 5 illustrates the reallocation procedure for the case of a receiver node reducing the scheduling priority. Figure 5(b) shows how the reduced scheduling priority increases the worst-case delay bound on N_p, which in this case is the receiver node. This is compensated by reducing the available slack s_p in the application.

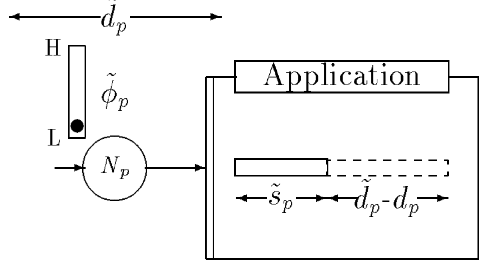
3.4 Resource Reallocation Protocol

We have described the resource reallocation procedure under the assumption that the necessary resources are available when the procedure is invoked. This must be insured by an appropriate procotol. In addition, the resource reallocation must abort properly in the case of failures or when resources unexpectedly become unavailable. We use a two-phase-commit protocol to ensure this atomicity of the reallocation process.

During the first phase, the participating nodes agree to tentatively reserve the required resources for the resource reallocation to take place. If some node does not



(a) Before Reallocation.



(b) After Reallocation.

Figure 5: Resource Reallocation on a Receiver Node.

have the necessary resources, it can deny to participate in the reallocation process, and thus force the reallocation to abort after the first phase. If the first phase completes successfully, all participating nodes have reserved the necessary resources, and the reallocation process can proceed.

During the second phase of the reallocation process, the actual reallocation is performed as described in the earlier sections. After the second phase, the newly allocated resources are committed to the channels. Since the reallocation process is confined to adjacent nodes, the communication overhead caused by this scheme is small.

4 Policies

The overall goal of the mechanisms described in the last section is to minimize the number of channel establishment requests that cannot be accepted because of lack of resources along the channel route (i.e. minimize the call-blocking probability). In the absence of a routing subsystem that has detailed knowledge about the location of resources, a good initial policy to minimize the call-blocking probability is to evenly distribute the available resources over the nodes in the network. We note that such “load-balancing” schemes in systems with multiple resource classes (bandwidth, scheduling priorities, buffers, etc.) on one side, and multi-metric performance requirements (delay, jitter, loss probability, etc.) on the other, is an area of research that needs much further investigation.

In this section we will use a very simple policy to illustrate, using an example, the use of the resource reallocation mechanisms described earlier. For simplicity of discussion, we assume that buffer space at the nodes is not a limiting factor, and therefore we balance the priority resource only. We define the *residual capacity* of a node as the amount of available resources at that node. The residual capacity directly reflects the blocking probability caused by the resource availability at the node. A balancing policy will therefore try to equalize the residual capacities of adjacent nodes by appropriately reallocate resources (in our case, priority resource). In a simple scheme, for example, a node N_p with small residual capacity initiates a resource reallocation round with an adjacent node N_s with larger residual capacity (or a set $\mathcal{N}_s$ of such nodes, if it is a branch node,) if, for an established channel C, ϕ_p is larger than ϕ_s. By decreasing ϕ_p while increasing ϕ_s, N_p and N_s agree on "migrating" priority resources from N_s to N_p.

The example in Figure 6 illustrates how such a simple policy can be used to replace resource relaxation by a dynamic resource reallocation scheme. In this example, the establishment process did not relax the resource allocations during the reverse pass of the establishment process; hence, resources are overallocated along the entire multicast channel. This results in (1) QoS guarantees at the receiver nodes (in this example slack) that exceed by far the performance requirements requested by the clients and, thus, (2) unduly low residual capacities at the nodes along the channel.

Immediately after the establishment of the channel, the receiver nodes trade in the excessive slack for reductions in scheduling priority. Since the application App_2 at node N_7 has more slack available hat App_1 on N_5, node N_7 can reduce the priority for the channel more aggressively.

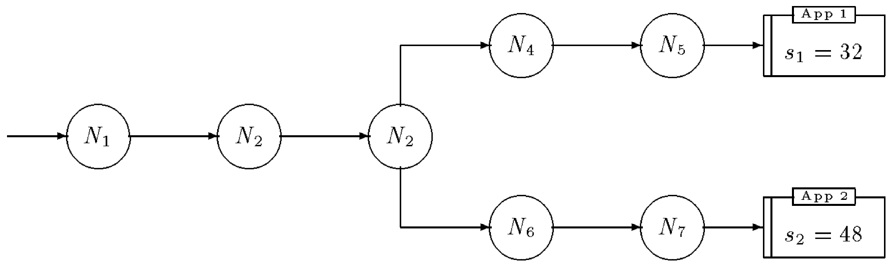
The priority reduction at the receiver nodes generates a gradient, which ripples back to the sender, eventually spreading out the resource requirements along the multicast channel. The node N_7 is not able to trade in all the slack in one step, even by reducing the priority ϕ_7 to zero. During later reallocations with its neighbor node N_6, node N_7 increases the priority again, and is able to trade in the rest of the slack. The result of this repeated resource reallocation process is a channel with fully relaxed resource allocations.

5 Conclusion and Future Work

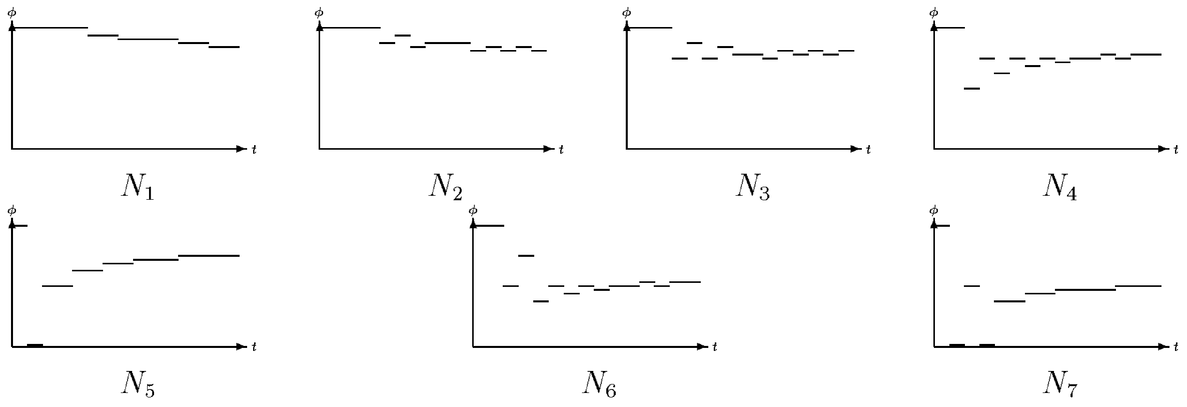
In this paper we have described Dynamic Resource Migration, a resource reallocation scheme for real-time connections in internetworks. The presented approach allows to reallocate resources, thus having them "migrate" between nodes, without affecting the timing guarantees provided to the clients by the active real-time connections.

The scheme is currently being implemented as a component of the control subsystem of Tenet Suite 2, the implementation of the Tenet Scheme 2 protocols for real-time multi-party communication. We are currently investigating the performance of simple gradient-based distributed load-balancing approaches in distributed real-time systems. This work ties in to the larger framework of load-balancing schemes for systems with multiple resource classes and multi-metric performance requirements.

The resource sharing schemes described in Section 2.2 complicate the design of reallocation policies further, since they introduce interdependencies between otherwise unrelated channels, if the channels are members of the same sharing group. We will explore combinations of local, gradient-based approaches with more global,



(a) Branch in Multicast Tree.



(b) Priority Levels as Result of Reallocations.

Figure 6: Dynamic Resource Reallocation as a Method for Resource Relaxation.

potentially off-line techniques to account for such interdependencies.

Acknowledgment

This research was supported by the National Science Foundation and the Defense Advanced Research Projects Agency (DARPA) under Cooperative Agreement NCR-8919038 with the Corporation for National Research Initiatives, by AT&T Bell Laboratories, Digital Equipment Corporation, Hitachi, Ltd., Mitsubishi Electric Research Laboratories, Pacific Bell, Tektronix Inc., and the International Computer Science Institute. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing official policies, either expressed or implied, of the U.S. Government or any of the sponsoring organizations.

References

[1] David Clark, Scott Shenker and Lixia Zhang. Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism.

In *Proceedings of ACM SIGCOMM'92*, pages 14–26, Baltimore, Maryland, August 1992.

[2] Domenico Ferrari, Anindo Banerjea and Hui Zhang. Network Support for Multimedia: A Discussion of the Tenet Approach. In Computer Network and ISDN Systems, pages 1267–1280, July 1994.

[3] Mark Moran and Riccardo Gusella. System Support for Efficient Dynamically-Configurable Multi-Party Interactive Multimedia Applications. In *Proceedings of the Third International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 143-156, San Diego, CA, November 1992.

[4] Abhay Kumar J. Parekh. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks. PhD Dissertation, Massachusetts Institute of Technology, February 1992.

[5] Craig Partridge and Stephen Pink. An Implementation of the Revised Internet Stream Protocol (ST-2). In *Journal of Internetworking Research and Experience*, pages 27-54, 1992.

[6] Jean Ramaekers and Giorgio Ventre. Client-Network Interaction in a Real-Time Communication Environment. In Proceedings of GLOBECOMM'92, pages 1140-1144, Orlando, Florida, December 1992.

[7] Lazar, A. and C. Pacifici. Control of Resources in Broadband Networks with Quality of Service Guarantees. In *IEEE Communication Magazine*, pages 66-73, October 1991.

[8] Bruce Mah. A Mechanism for Administration of Real-Time Channels. Tech. Report UCB/CSD-93-735, University of California, Berkeley, CA, March 1993.

[9] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker and Daniel Zappala. RSVP: A New Resource Reservation Protocol. In *IEEE Networks Magazine*, Vol. 31, No. 9, pages 8-18, September 1993.

[10] Riccardo Bettati, Domenico Ferrari, Amit Gupta, Wendy Heffner, Winnie Howe, Mark Moran, Quyen Nguyen and Raj 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, NH, April 1995.

[11] Scott Shenker and Lee Breslau. Two Aspects of Reservation Establishment. In *Proceedings of SIGCOMM'95*, Cambridge, MA, August 1995.

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

[13] Amit Gupta and Mark Moran. Channel Groups: A Unifying abstraction for specifying inter-stream relationships. Tech. Report TR-93-015, International Computer Science Institute, Berkeley, CA, March 1993.

[14] Colin Parris, Hui Zhang, and Domenico Ferrari. Dynamic Management of Guaranteed Performance Multimedia Connections. To appear in *ACM Journal of Multimedia Systems.*

[15] Hui Zhang and Domenico Ferrari. Rate-Controlled Static Priority Queueing. In *Proceedings of IEEE INFOCOM'93*, pages 539-546, San Francisco, California, April 1993.

[16] Domenico Ferrari. Design and Applications of a Delay Jitter Control Scheme for Packet-Switching Internetworks. In *Computer Communications* 15(6):367-373, July-August 1992.

[17] S. Jamaloddin Golestani. A Stop-and-Go Queueing Framework for Congestion Management. In em Proceedings of ACM SIGCOMM'90, pages 8-18, Philadelphia, Pennsylvania, September 1990.

[18] Charles R. Kalmanek, Hemant Kanakia, and Srinivasan Keshav. Rate Controlled Servers for Very High-Speed Networks. In *IEEE Global Telecommunications Conference*, pages 300.3.1-300.3.9, San Diego, California, December 1990.