INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center St. • Suite 600 • Berkeley, California 94704-1198 • (510) 643-9153 • FAX (510) 643-7684

Fault handling for multi-party real-time communication

Amit Gupta and Kurt Rothermel[*]
{amit,rothermel}@icsi.berkeley.edu

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

TR-95-059

October 1995

Abstract

For real-time communication services to achieve widespread usage, it is important that the network services behave gracefully if any component(s) fail. While other researchers have previously considered failure-handling for non-real-time communication as well as for unicast real-time communication, these failure-recovery techniques must be reexamined in the light of the changes introduced by the new protocols and services for supporting multi-party real-time communication. In this report, we describe techniques and mechanisms for maintaining network services for multi-party real-time communication in the face of failures that may make parts of the network inaccessible. The key goal is that the protocols should provide high performance in the common case (i.e., in absence of failed components) and the network performance should gracefully degrade in face of network failures; e.g., in the presence of network faults, the routes selected may not be as good, the connection set-up may take a little more time, or resource allocation may be less efficient. We describe appropriate policies for storing state in the network, as well as the mechanisms for re-establishing connectivity for previously established connections and to permit setting up new connections to existing conferences. We also describe a redundancy-based approach, using forward error correction (FEC), and dispersing the FEC'ed data among disjoint routes. With these mechanisms, we can make multi-party real-time communication protocols robust to single and/or multiple failures in the network, *without* diluting the

[*]Now with University of Stuttgart, Institute of Parallel and Distributed High-Performance Systems(IPVR), Breitwiesenstr. 20-22, D-70565 Stuttgart, Germany

strength of the performance guarantees offered, or sacrifing the system performance in the common case, i.e., when all components work correctly.

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 [8, 17, 30, 31, 36, 37], 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 [8, 17, 26].

While failure handling and recovery for computer networks has been well-researched for non-real-time communication, there is very little research in handling failure in case of real-time communication. Also, the proposed solutions suffers from one or more of several drawbacks:

- They may assume a very limited communication model. For example, [4, 34, 33, 35, 32] limit the treatment to simplex, unicast channels.
- They may dilute the level of performance guarantees. For example, RSVP[25, 29, 28, 42] uses a soft-state mechanism that provides a "best-effort" approach to failure-handling. The failure-handling can be significantly delayed because of the soft-state nature, and there are no mechanisms by which an end-user can determine if the network is still supporting the performance that it requested, or if, due to the failure re-routing algorithm followed, the required resources were not available.
- They may design and optimize the system for handling failures [42, 6, 8, 7]. The common-case (i.e., with all components working correctly) performance may suffer as a consequence.

For real-time communication services to achieve widespread usage, it is important that the schemes and protocols allow the network's management to provide a useful and usable service; since all networks can fail (for example, due to operator error, due to natural calamities, like earthquakes, or due to hardware and software crashes), it is important that we incorporate useful failure-handling techniques in designing multi-party real-time communication protocols.

This paper describes techniques that can be used to restore guaranteed performance multicast channels, using the available unused network capacity, after single and/or multiple link and node faults[1]. We assume a fail-stop model of system faults

[1]A system fails when its behavior deviates from its specified behavior and it therefore can not provide the desired service. An error is that part of the system state which, unless appropriately corrected, is liable to lead to subsequent failure. The cause of an error is a fault. In this discussion,

and failures [39, 27, 41]. Also, though our algorithms are optimized for high performance under the common case that the components work correctly (as the failures are rare events), the network should be able to gracefully handle multiple failures.

The paper is organized as follows. Section 2 describes the salient features of the Tenet multi-party communication protocols that provide the background for the research described here. In Section 3, we discuss some of the key ideas and considerations that motivated us to the current design. Failure-handling mechanisms encompass two related issue: maintaining distributed state in presence of failures in the network, which we discuss in Section 4; and recovering multicast trees from node and link failures, which we discuss in Section 5. We describe a hybrid mechanism in Section 6 and then conclude this paper with a brief review of related work in Section 7.

2 Tenet protocols for multi-party real-time communication

In this paper, we illustrate our approach to failure-handling in the framework of the Tenet multi-party real-time communication protocols [17]; 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 [5] describes the relevant aspects of the Tenet protocols; in this discussion, we limit ourselves to discussing a few of their 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 handling network failures in the Tenet Suite 2.

2.1 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 in the set. A real-time channel ("channel" for short in the sequel) 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

we will sometimes use these terms interchangeably, as long as the appropriate meaning is clear from the context.

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 subtree vertex 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 (where a server is a node or a link) reached by a copy of the message; if any of the tests is unsuccessful, a *channel-reject* message is sent back toward 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 setup 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 vertex, where they are merged with them 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 replicas 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. 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 all this activity, the source receives the results of the request within a single return message, and transmits them to the client for evaluation and further action.

The channel so created, reaching the destinations that have accepted the request, is now ready for immediate use and remains in existence until it is torn down explicitly by the client or destroyed by an irrecoverable failure.

2.2 Multi-party issues in real-time communication

In this section, we discuss some of the issues in the design of network services for multi-party communication; these issues arise primarily due to the multi-party nature of the communication (e.g. multicasting, dynamic changes in the conference membership, resource sharing etc. as described below).

Multicast groups: 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.

Now, 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) know 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 support 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 in their ability to handle the data, and the QoS requirements that 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 very large conferences.

Also, multi-party conferences tend to be long-lived; the presence of multiple senders and receivers raises another issue: the membership in a multicast group may be dynamic, i.e., receivers may join (or leave) to listen to a session while it is in progress. It is important that the network service provide support for dynamic changes in group membership.

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.

Resource sharing: Traditional real-time network systems (e.g., [17]) 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 [22]. 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 speak 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). Unfortunately, as per the traditional approach, if fifty of these channels overlap along some common sub-path, the network would reserve enough resources for fifty audio channels; this is clearly wasteful over-allocation. 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 [23]; 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.
Advance reservations: Conferencing and other important distributed multi-party multimedia applications would benefit from a network service the provides support for advance reservations. The network service clients who wish to set up multimedia multi-party meetings need to schedule those meetings in advance to make sure that the participants will be able to attend, and would like to obtain assurances that the network connections and the other required resources will be available for the entire duration of the meeting. The Tenet Scheme 2 provides its users with the ability to book network resources (far) in advance of their use[19, 21]; this advance booking requires long-lived state in the network and it thus raises some interesting questions. How is this state stored? If a link goes down, should we also reroute the advance-reserved channels that are to traverse this link in the distant future? Do we need separate mechanisms for handling advance reserved channels or can we effectively re-use mechanisms designed for non-advance channels?

2.3 Tenet Suite 2: State information management and channel establishment

We can now describe the two key components of the connection establishment and management system for Real-time Channel Administration Protocol (RCAP) for the Tenet Protocols Suite 2: state information management, and channel establishment.

In the Tenet Suite 2, the channel administration system consists of a set of RCAP daemons, one on each node in the network, which communicate with each other via reliable messages (using TCP connections). In this object-oriented design, the RCAP tasks are carried out by the different objects in these RCAP daemons; some of these objects store state information for managing multicast channels and groups, some others perform admission tests and store resource allocation information, and others provide routing and access control functionality. We now describe, as per the current Suite 2 design and implementation, the objects that participate in state information management and in channel establishment.

- **Target Set Object:** In Suite 2, there exists one Target Set object for every Target Set (and thus, for each "conference session"). The conference organizer requests the network to *create* a Target Set; then the destinations *join* the Target

Set and the senders *establish* channels to the Target Set. To join a Target Set, the destination client sends a *Join-message* to the Target Set object. When the Target Set object receives the *Join-message*, it stores the destination address and the requested performance information; it also sends a message to the already established channels to add this new destination to their destination list.

- **Channel Object:** In Suite 2, there exists one Channel object for every channel. As mentioned before, a channel sends data to members of a Target Set. When a channel is created, the client also specifies the Target Set whose members will receive the channel; in the current design, mainly due to efficiency considerations, the Channel Object is co-located with the corresponding Target Set object (i.e. at the same RCAP daemon/node). When a client wants to establish a channel, it send an *Establish-message* to the corresponding Channel Object; when a channel object receives an *Establish-message* , it obtains the Target Set membership list from the Target Set object; it then sends this information to the routing system, through the Routing Stub object, to obtain the multicast route. It then sends an *Establishment-request* to the RCAP daemon at the channel source for channel establishment.

- **Group Object:** There exists one Group Object (also referred to as Sharing Group object) for every *resource sharing group*; this group object maintains information about the aggregate traffic specification for that sharing group.

- **Establishment Object:** There exists one Establishment object at each RCAP daemon. The Channel object sends the *Establishment-request* message to the Establishment object at the channel source; the Establishment object contacts the Local Resource Manager (LRM) to allocate resources (the LRM also decides whether resource sharing can be used to reduce resource allocation, or to accept new channels without allocating any more resources). The LRM returns the local performance bounds to the Establishment object, which forwards the *Establishment-request* message to downstream nodes; this establishment process proceeds in the manner specified in Section 2.1.

- **Routing Stub Object:** The RCAP daemon talks with the routing system through the Routing Stub (RS) object; there is one such object at each RCAP daemon. The RS object talks with the Routing Managers that form the distributed routing system. The routing system maintains a fair amount of state in this system. First, for all established channels, the routing system keeps information about resource allocations and local performance bounds at all nodes along the (multicast) channel route; this information is used for obtaining incremental routes for supporting dynamic changes to the Target set membership. Second, the routing system tries to maximize resource sharing gains by increasing the common sub-path overlap for channels that can share resources. For doing this, the routing system associates, with each Sharing Group, a "network model"; the network model keeps information about the links traversed by the channels belonging to that group. This also adds to the state information kept by the routing system.

3 Key ideas and considerations

In this section, we describe the key ideas and considerations that motivated the design of out failure-handling techniques and mechanisms. It is important that we distinguish between the design objectives and goals; the objective is to provide performance guarantees for multi-party real-time communication; the goal is to handle failures in the system. It is not acceptable that we *soften* the performance guarantees to provide failure-handling. It is acceptable that if a network component fails, then some channels may not be able to recover to the previous level of Quality of Service; or if data is lost for some reasonably small transition time; or if the network is partitioned, some channels may not be able to recover from the consequent failures. However, failure-handling techniques should not force us to dilute the performance guarantees that we can offer to the clients when the network does not suffer from failures.

In our approach to designing failure-handling techniques, a key component is failure-detection; in our approach, failures are explicitly discovered; this discovery may result from either (a) a time-out occurring when a node contacts another to obtain information or to carry out some tasks, or (b) as a result of a *monitoring* process, under which all nodes periodically send messages to each other via TCP; nodes reports failures to the network if the corresponding TCP connection is not established. This discovery triggers the failure-handling mechanisms as well as the fault-handling mechanisms that attempt to recover the network from the corresponding fault(s).

Two key ideas comprise our approach to designing failure-handling techniques for multi-party real-time communication: fate-sharing, and the principle of optimality.

3.1 Fate-sharing

The fate-sharing principle was first described in [7]. The fate-sharing principle says that it is acceptable to lose the state information associated with an entity if the entity itself is lost at the same time; this principle can be used to make decisions about the nodes where state information should be placed. For example, a channel can not exist (or is useless) if the source node goes down; in this case, the fate-sharing principle states that the critical state information associated with a channel should be available (or reconstructible) at the channel source node. It should be noted that this information may also be present at other nodes in the network (for performance reasons, for example); the principle just requires that it be available at the source node. Also, the fate-sharing principle does not rule out connection-oriented services; it merely states that the connection state information should be available at the source node.

3.2 Principle of optimality

An important principle is that we should optimize the design for the common case. While the design should be robust to multiple failures in the network and it is desirable that the performance should degrade gracefully in the presence of failures, the common case for network operation is the "no-failure" case; therefore, the protocols should be designed for the highest performance if there are no failures in the network.

This principle implies that the network protocols may be designed, either explicitly or implicitly, to operate in two different modes: the "normal" (failure-free) mode and the "failure" (in presence of failure(s)) mode. In the failure-free mode, the techniques are chosen for high performance. However, when network components fail, the network switches to *failure-mode* operation; in this mode, the techniques chosen may not perform as well as the *normal-mode* techniques. These techniques would, though, work well in a robust manner. Of course, we still need to make sure that the *normal mode* operations respond to network failures by switching to the *failure mode* techniques in a robust manner.

4 Managing state information in presence of faults

As we mentioned in Section 1, protocols for providing performance guarantees must keep some state in the network; this state keeps track of the resources already allocated to existing channels. Also, multi-party real-time communication requires the network to maintain state information about the multicast groups, the group members and their performance requirements, the existing channels, and the inter-relationships among these channels. In Section 2, we described how some of this information is maintained in the Tenet Suite 2; we also described, in Section 3, the fate-sharing principle. In this section, we describe how the fate-sharing principle can be used in setting policies for distributing the state information across the network nodes; this distribution improves the accessibility of the desired state information in the presence of faults in the network.

- **Target Set:** The Suite 2 implementation of Target Sets (a single, non-distributed object maintaining all state information for a given Target Set) is clearly susceptible to failures in the network; if the node with the Target Set object goes down, then no further connectivity for that Target Set is possible; new channels can not be established because the corresponding Channel Objects will not be able to obtain Target Set membership, and new destinations can not join a Target Set because the corresponding Target Set object is not available. This is clearly bad news.

  The good news is that the problem is reasonably easy to rectify; the basic solution follows from the fate-sharing principle. The fate-sharing principle would suggest that the Target Set membership information (e.g., destination performance parameters) be maintained with the corresponding destination nodes. In this case, the performance parameters for a given Target Set member will be available unless that destination node goes down; if a node goes down, then we do not need the performance parameters for the members at that node.

The optimality principle would dictate that for *normal-mode* operations, we should be able to obtain this information without necessarily querying all destinations; this leads us to the second component of our solution. The (possibly distributed) Target Set object maintains a "cache"; as long as the Target Set object is available, it maintains correct, up-to-date information about the Target Set membership[2]; if the Target Set object becomes unavailable, the Target Set members re-create the "cache" by creating a new copy of the Target Set object, and filling in the state information about the members' performance requirements. Reliable, non-real-time multicast group management protocols can be used to significantly simplify this task; For example, if IGMP is used, the network can associate each Target Set with an IP HostGroup; all Target Set members also listen to this HostGroup for requests to provide the requisite state information. If the "cache" becomes unavailable (for example, due to a network partition) the cache can be recreated using standard distributed leader election protocols[1, 38].

- **Channel object:** In the current Suite 2 implementation, the Channel object is "co-located" with the Target Set object, i.e., the Channel Object for a channel resides on the same network node that the Target Set object for the corresponding Target Set resides. By the fate-sharing principle, the Channel Object should be located at the channel source; this will ensure that the channel state information is available as long as the channel source is available. We do not need any other changes for maintaining the channel state.

- **Resource sharing:** Resource sharing information is stored at the corresponding Sharing Group object. When a member channel is established, state is also installed at the nodes along the channel route; the routing system also maintains state required to maximize the common sub-path overlap among channels that can share resource allocations.

  We do not need to make any changes for the resource sharing information. If the sharing-related routing information is not available, the routing system can still proceed; the routes generated may not be as good in optimizing overlaps among member channels, but wherever these paths overlap, resource sharing can be used to improve resource allocation efficiency. This follows the principle that if some components fail, the performance may degrade but the protocols will still work correctly.

  If the Sharing Group object is not available, the channel establishment can still proceed. If this SG object was available when another member channel was

[2]State information about the members' performance parameters is thus duplicated

previously established, the resource sharing information will be available along that channel's path; this information can be used to share resource allocations on common sub-paths. If the SG object becomes available later, the member channels then established can share resource allocations with each other as well as with channels previously established when the resource sharing information was not available. Even if the SG object becomes permanently unavailable soon after the initial creation, the protocols will work correctly; they will not be able to use resource sharing to improve resource allocation efficiency. The principle of optimality applies again.

- **Routing:**

  As we mentioned in Section 2, the routing system maintains, in addition to the network-wide real-time load information required for setting up routing tables, state information for already established channels: (a) the established route and resource allocation for the current channel at each node along the channel route; this information is used for incremental establishment for adding new destinations which dynamically join the Target Set after that channel has already been established; and (b) for each sharing group, the nodes and the links in the network where resources have already been allocated, either for the entire group or for some member channels.

  As per fate-sharing principle, it would be useful to store the established route and resource allocation information at the corresponding channel source; this copy is then available as long as the channel source is available.

- **Advance reservations:**

  As mentioned in Section 2.2, the Tenet Protocol Suite 2 supports advance reservation of network resources; these mechanisms are based on *resource partitioning*[18, 20]. The connection establishment signaling is the same for advance channels as for non-advance reserved channels, with the only difference being in the the Local Resource Managers (LRMs); these changes are described in [19, 21].

  The resource allocation information, and the associated state, is long-lived for advance-reserved channels; there are two alternatives for storing this information: the simple alternative is to store the information at the same node(s) as where this information would be stored for non-advance channels; the other alternative is to store the information at other "central" and "more robust" nodes. The second alternative requires more work, to maintain information about these "robust" sites, and for protocols for moving information to and from these sites. On the other hand, the first alternative is simpler, and it also maintains uniformity with the state storage and failure recovery for non-advance channels. Due to these factors, we chose the first alternative. Another important concern relates to the time when recovery should be attempted. First, the network usually has a lot more time for recovering advance-reserved channels. Second, failure-recovery may not be necessary if fault-repair (for example, detecting and replacing the faulty components) can reasonably be expected to finish before the advance-reserved channel's lifetime, it makes sense to not do any failure recovery at all.

5 Recovering multicast trees from node and link failures

In the previous section, we described the mechanisms for maintaining the state information in a robust, fault-tolerant manner. In this section, we describe the mechanisms that can used to maintain and recover multicast transmission from failures in the network. We present two approaches: (a) when the network detects a failure, it can try to "repair" and "re-build" multicast trees to recover from link and node failures, or (b) we can redundantly transmit data on multiple channels. These approaches are not mutually exclusive; these can be used together to provide better tolerance to failures in fault-prone networks.

5.1 Repairing existing multicast trees

We use the standard tree terminology in describing these mechanisms: root , predecessor, successors, and descendants (the reflexive transitive closure of the successor relationship). Also, we limit this discussion to mechanisms that recover one single channel from an already detected failure[3]. [4, 34] describe some interesting interactions when the network attempts to repair several unicast channels in parallel, and we expect similar interactions with multicast trees. However, such discussion is outside the scope of this report; we plan to investigate these interactions in the near future.

Due to link and system failures, a channel's multicast tree may shrink or even may be split in a number of subtrees. A node in a channel's multicast tree is called *destination orphaned* (or d-orphaned for short) if there is no destination in this node's set of descendants. A node is defined to be *sender orphaned* (or s-orphaned for short) if the root of this node's subtree is not the sender, i.e. there exists no path from the sender to this node. Figure 1 and Figure 2 illustrate these concepts. We also assume that control messages between nodes are transferred over reliable connections; for example, in the Tenet Suite 2, the nodes exchange messages over TCP . These connections guarantee ordered, unicast message delivery in the absence of link and system failures.

5.1.1 Recovery for D-orphaned nodes

At every node in the network, the RCAP daemon maintains, for each channel that passes through that node, the list of the node's successors (i.e., the node's descendants in the multicast tree) ; this list is called the *successor-list* for that channel. A node in a channel's multicast tree becomes d-orphaned if all of its successors are unavailable

[3]Many such recovery processes can proceed in parallel, independent of each other

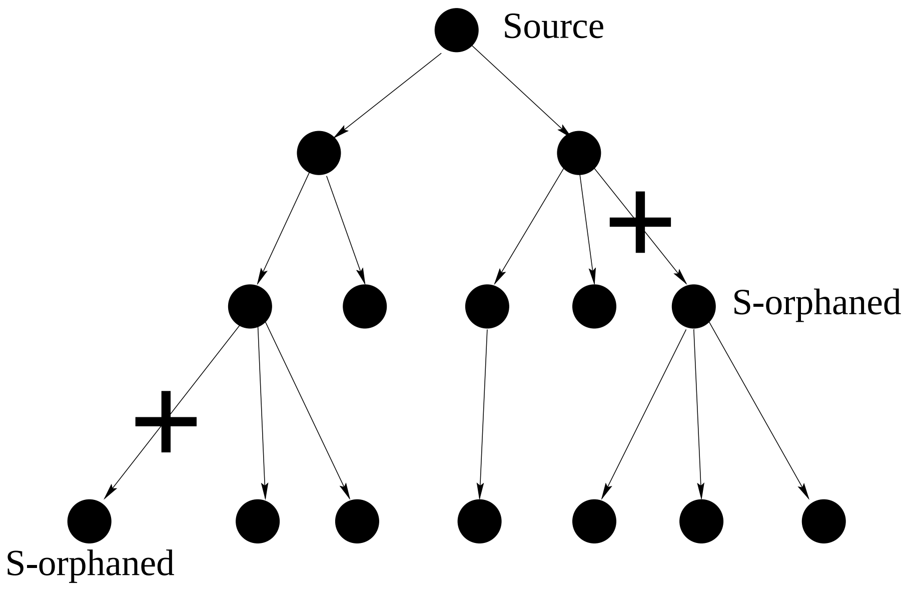

Figure 1: Example: s-orphaned nodes

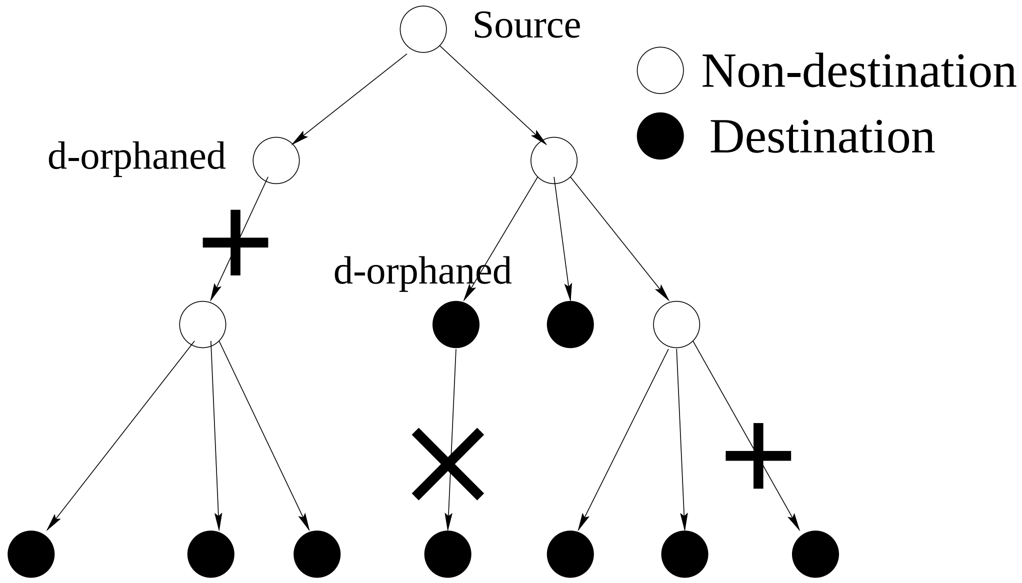

Figure 2: Example: d-orphaned nodes

or become d-orphaned themselves; the sole exception is that a destination node can never be d-orphaned.

When a node learns (for example, through the monitoring process) that its neighbor has become unavailable, it removes this neighbor from all the *successor-lists* that the neighbor is on; it does the same when a *d-orphaned* message arrives from a neighbor. In either case, the node should release the resources allocated at the outgoing link to that neighbor. Also, a non-destination node becomes d-orphaned when its *successor-list* becomes empty (and a destination node is never d-orphaned).

When a node becomes d-orphaned, the failure-recovery mechanisms at that node should release the resources locally allocated to that channel. Also, the node should send a *d-orphaned* message to its predecessor. The node can then remove all local state information about that channel. With these steps, the node completes the recovery for the d-orphaned state.

5.1.2 Recovery for S-orphaned nodes

A node becomes s-orphaned for a channel when either its parent becomes s-orphaned or it can no longer get data from its parent. When a node becomes s-orphaned, all its descendants also becomes s-orphaned. Fault-recovery for d-orphaned nodes is simpler because it only requires reclaiming the allocated resources; on the other hand, failure-recovery for s-orphaned nodes is more difficult because it requires routing and allocating resources along a new "repair" path from the channel source. A key issue in such failure-recovery is selecting the node which this new path will connect the channel source to. We have a spectrum of choices for this selection: at one extreme, we can select, for this failure-recovery, all destination nodes that are s-orphaned; at the other extreme, for this failure-recovery, we can select the "root" s-orphaned node, i.e., the s-orphaned node whose link to its parent failed.

- When the root s-orphaned node learns (again, for example, through the monitoring process) that its parent has become unavailable, it sends a message to all destinations in its sub-hierarchy, on receipt of which each destination independently attempts to set up a path from the source to itself.

- When the root s-orphaned node learns that its parent has become unavailable, it declares itself s-orphaned and tries to reconnect its entire sub-hierarchy by re-establishing the channel along a new path from the source to itself.

As usual, there are pros and cons for these different policies. The advantage of reconnecting sub-hierarchies is the efficiency of the entire rejoin procedure. Instead of individually re-connecting each each destination in the disconnected sub-hierarchy, the network attempts to re-connect only the root of the sub-hierarchy. In reasonably well-designed networks under reasonable operating load, this procedure should succeed in most cases. If it fails, the network can delegate the re-connection responsibility immediately to the destinations in the sub-hierarchy. Consequently, even in the failure case, only one extra re-connection operation has to be performed. The advantage of the second policy, which immediately transfers the re-connection responsibility to the destinations in the sub-hierarchy, is that one might end up with better routes and a more balanced network load.

There are several alternatives that can be examined as policy issues. A critical issue relates to the steps that the failure-handling system should take if the first attempt fails due to non-availability of resources. We will discuss these issues in Section 6.

5.2 Redundancy based approaches

A simple, albeit expensive, solution would be to redundantly transmit data on two separate, preferably disjoint channels; this would ensure tolerance to a single failure in the network, though it would cost twice as much in the network resource allocation as well as in data transmitted. Such redundancy based approaches are appropriate in a network in which either the transmission capacity is relatively abundant, or for applications where tolerance to failure is critical (tele-surgery?).

More generally, instead of directly sending multiple redundant copies of data on separate channels, we could employ Forward Error Correction (FEC) techniques along with multiple-channel reservations; for example, the can use FEC to send data on five separate channels, where the data sent on any four channels would be enough to reconstruct the original data stream. In this case, the application performs the key failure-management related tasks, including FEC coding and decoding, as well as creating additional redundant channels if increased protection is required, or if one of the existing redundant channel fails due to a network failure. What is the network's role in this case?

All this application-level work would be quite useless without appropriate network support. For example, with the current Internet routing protocols ([9, 2]), all channels would follow the same multicast route to the respective destinations; thus, if any node or link along this route was to fail, all the channels would fail together and this redundancy would be absolutely useless.

Thus, for redundancy-based approaches to work, the network must provide additional support; in this case, the network should try its best to route these channels along mutually disjoint routes. For such disjoint routing to work, the client has to inform the network that these different channels are all due to the same FEC'ed data stream. The clients can use the *channel groups* [23] to pass this information to the network service. The network can then use this information to obtain mutually disjoint routes.

We now present a simple example to illustrate this mechanism. Here, the application creates five channels, say C1, C2, C3, C4, and C5, to transmit FEC'ed data belonging to a single data stream. In this case, one sequence of calls can be:

- Create a Target Set, say TS. Also create a Channel Group, say CG, with *disjoint routing* relationship.

- Create channels C1, C2, C3, C4, and C5.
- Include these five channels (C1 ... C5) in the group CG.
- Request the network to establish these five channels (C1 ... C5).

When the network gets the establishment request, it knows that these five channels are related in that as far as possible, they should be provided mutually disjoint routes.

6 A hybrid mechanism

We now discuss the alternative policies that the network can try if the initial failure-recovery repair fails due to non-availability of network resources. If the initial re-connection is attempted to the destination nodes directly, and that attempt fails, that re-connection would probably fail again due to non-availability of network resources[4]. On the other hand, if the root s-orphaned node fails in re-connecting itself, it is more likely that re-connection to one of its descendants may succeed. In this case, the root s-orphaned node can either (a) delegate this re-connection responsibility to its immediate successors, (b) delegate the re-connection responsibility to the destination nodes in its sub-tree, or (c) delegate the responsibility to the descendants at a certain level in-between.

Also, after a (small) number of repeated re-connection attempts, the network should give up on re-connecting these destinations and release all the allocated resources accordingly.

One can expect that trade-offs among these different alternatives depend on the network load and the properties of the multicast routing system. If the routing system has enough information, it can suggest the policy that should be followed. For example, during the initial re-connection attempt, the routing system could decide - based on its information on the current network conditions - that the best fall-back would be for the root s-orphaned node to immediately delegate the re-connection responsibility to the final destinations.

We now describe a simple mechanism that can support the above-mentioned alternatives; the appropriate policy can be chosen, and dynamically re-selected, by modifying the relevant variables. Assume that each node maintains two variables:

- **StartLevel**: This variable specifies on which (relative) tree level the reconnection process starts. Assume that node N becomes s-orphaned due to the unavailability of its predecessor node. Then StartLevel defines, where reconnection starts relative to N. If StartLevel equals 0, then reconnection starts at N; if StartLevel equals 1, then reconnection starts at N's successors. If the

[4]A repeated re-connection attempt can succeed either because the newer route is better than the previously attempted one, or because in the meantime, some other channel released its resource allocation. Neither is likely in the short repair time frame that we are interested in, especially as we expect our routing system to provide a reasonably good route for the re-connection attempt.

reconnection is to directly start at the destinations, the StartLevel is set to *infinity*.

- **RetryLevel**: If re-connection attempt fails at node N, then RetryLevel specifies on which tree level relative to N at which reconnection should be re-attempted. For example, if RetryLevel is set to 0, then N re-attempts reconnection, after a small delay; if RetryLevel equals 1, then N's successors retry. If RetryLevel is set to *infinity* then only the descending destinations retry the reconnection.

Both variables can be set in the initial RCAP daemon configuration and can be set and dynamically adapted by network management, in coordination with the routing system, based on the feedback obtained from previous rerouting attempts.

In the following description, we assume that an *s-orphaned* message includes the globally unique Channel Id along with "Level" information. When a node, say N, learns that it has lost connectivity to its predecessor, it checks the local StartLevel variable. If StartLevel equals 0, it immediately requests a connection from the source to itself. Otherwise, an s-orphaned message is sent to N's successors, with *Level* set to (StartLevel - 1). After sending this message, N releases all resources and state information associated with the corresponding channel.

When a node receives an s-orphaned message it checks whether the channel indicated in the received message is (locally) still active. If the channel is not active anymore, it concludes that this channel has suffered from multiple network failures and other failure recovery is already in progress and it can safely ignore the current *s-orphaned* message. Otherwise, its actions depend on whether it is a destination node for the current multicast.

In the case the receiver is a non-destination node, it checks "Level" included in the message. If Level equals 0, the node attempts re-connecting the channel source to itself; else, it decreases Level by 1 and propagates the s-orphaned message with the decremented Level value to all of its successors. After sending this message, the receiver releases all resources and state information associated with the corresponding channel.

If the receiving node is a destination, the node always attempts to re-connect the channel source to itself. independent of the Level value included in the received message. Also, if the node has any successors and the Level value is greater than 0, the Level is decremented by one and included in the s-orphaned message(s) sent to the current node's successors.

If the re-connection attempt issued by node M fails, the node's activity depends on whether or not the node is a destination. If M is a destination, it periodically tries re-connecting the channel source to itself till the re-connection procedure either succeeds or returns the information that the channel does not exist anymore (or some threshold on the number of unsuccessful trials is reached). To avoid that an s-orphaned message is sent several times to its successors, N sets a flag in the local channel state information when sending this message the first time.

If M has any successors, it checks its RetryLevel value. If RetryLevel equals 0, then M tries re-connecting the channel source to itself. Otherwise, it sets "Level" in an s-orphaned message to (RetryLevel - 1) and sends this message to its successors. After sending this message, M releases all resources and state information associated with the corresponding channel.

By appropriately setting *StartLevel* and RetryLevel, we can support a wide variety of policies.

7 Related work

Fault-handling has been extensively investigated in telecommunication networks. [16] discusses fault-recovery in telecommunication networks against a layered model: the highest is the *switched layer* where the unit of communication and fault-recovery is the call and the recovery action is to simply update the routing tables around the faults, with existing calls redialed by the end-systems; the next layer is the *cross-connect* layer where the units of communication and recovery are *trunks* (e.g., T1 or T3) where the recovery action is to reroute remaining trunks around the faults so as to meet the bandwidth requirements of the flows and the recovery mechanisms are based on pre-computation, and running dynamic distributed algorithms[4, 34, 3].

The ST family of protocols (ST, ST-II, ST2+) provide connection-oriented multicast-based real-time communication service [24, 11, 12, 13, 14, 15, 36, 40]; the latest protocol specification (ST2+) is described in [10]. The ST2+ protocol does not directly support any Multicast Group abstraction (IP multicasting has the HostGroup abstraction, the Tenet multi-party protocols have the Target Set abstraction); any such abstraction must be supported by higher-layer protocols and thus any recovery for such Multicast Group information is outside the scope of ST2+. Similarly, ST2+ expects the "application" to provide the *Sharing Group* information each time a channel is to be established; it does not maintain such information itself. Consequently, ST2+ is not responsible for recovering such information; this simplifies ST2+ protocol significantly but adds to the overhead in the higher-layer protocols. For detecting link and node failures, the ST2+ agents (similar to RCAP daemons) periodically exchange "HELLO" messages; if a timeout occurs at an ST2+ agent, it infers that either the next node is down or the link has failed and then it initiates the recovery mechanisms.

For Tenet Protocol Suite 1, [4, 34, 3] describe fault-handling and recovery mechanisms for simplex, unicast channels with performance guarantees. The unicast nature of communication considerably simplified their task; they did not have to consider multiple destinations; nor did they have any multicast group or resource sharing group information to recover. They proposed techniques for rerouting simplex connections around failed links, while maintaining the performance guarantees promised to these connections. They contrasted the performance of local rerouting techniques (where the nodes adjacent to the faulty node try to reroute the connection) with that of global rerouting techniques (where the new route is computed from the channel source to the channel destination), and also proposed intermediate, "hybrid" techniques. When a node or a link fails, the system tries to recover, in parallel, the different channels that traverse the failed component; these authors also observed interactions among these different parallel recovery attempts. The different channel recovery processes compete for the same set of resources, resulting in poor system performance (number of successfully rerouted connections). These experiments also showed that delaying some of these recovery processes can lead to better overall system performance.

When the DARPA Internet protocols were designed, the most important goal was that the Internet should continue to supply communication service even when the networks and gateways were failing. The *fate-sharing* principle was extensively used to protect the network from multiple intermediate failures. The network layer protocol (IP) was designed to provide a basic datagram-oriented service on top of which multiple protocols could co-exist to provide a variety of services, both reliable and unreliable. The reliable transport layer protocol (TCP) was designed to store all connection-related state at the end-nodes; also, each packet carries all information required to route it to the destination. As no network state is stored at any intermediate nodes and because the underlying network protocol (IP) transparently re-routes the datagram in the presence of network faults, the TCP/IP set provides reliable communication service in presence of multiple network faults. Of course, this task is considerably simplified because these protocols do not provide any performance guarantees.

For providing performance guarantees, the IETF researchers designed Resource reSerVation Protocol (RSVP) [42]. The fate-sharing principle is also used extensively in RSVP design; RSVP also uses a *soft-state* approach to make the protocol robust against network faults. In this soft-state approach, a time-out is associated with all state information stored at intermediate nodes; the end-nodes must periodically refresh the state information for long-lived sessions. The RSVP designers limited the protocol to resource reservation state set-up only; the other issues, including routing, packet scheduling disciplines, and admission control, are handled by other protocols. In the current implementation, RSVP works on top of IP multicasting; IGMP maintains the multicast group membership and the IP multicasting also provides multicast routing. When a new sender wants to set up a flow[5] to a multicast group, its sends a "PATH message" to that multicast group and starts sending data. All destination (members of a multicast group) reply to the PATH message by sending a "RESERV message" back to the senders[6]. In this soft-state system, the PATH and the RESERV messages are periodically re-sent; when an intermediate node receives any such message, it stores (or refreshed) the relevant state information, along with the associated time-out. IP multicast routing transparently reroutes the messages around any faults in the network; the state is re-established along any new sub-paths.

The key difference between the RSVP approach and the approach the we described is that RSVP is designed primarily to operate well in the presence of multiple failures;

[5]The term "flow" is roughly equivalent to the term channel as used in this paper.

[6]The traffic overhead is reduced by *merging* the different messages that belong to the same multicast group.

we designed our protocols to provide highest possible performance in the absence of failures, and yet providing correct service, with possibly degraded performance, when some components fail. For soft-state protocols, the network management must set the refresh interval carefully: if the refresh interval is too small, the traffic overhead for the messages becomes excessively high; if this interval is too high, the network will take a lot of time in reacting to failures, and it can also result in significantly higher losses in network resource allocation along old, stale sub-paths when the flow is rerouted. With these soft-state mechanisms, failure-handling becomes completely transparent; however, it is possible that the failure-handling mechanisms' performance may be degraded due to the same synchronization and competition phenomena as observed in [3, 4, 34]. Also, the clients have no way of determining if they are indeed receiving the performance guarantees that they requested.

8 Summary

We have presented a fully-distributed scheme for maintaining network services for multi-party real-time communication in the face of failures that may make parts of the network inaccessible. In our scheme design, the key goal was that the protocols should provide high performance in the common case and the network performance should gracefully degrade in face of network failures; e.g., in the presence of network faults, the routes selected may not be as good, the connection set-up may take a little more time, or resource allocation may be less efficient. We described the *fate-sharing* based policies for storing state in the network, as well as the mechanisms for the network to re-establish connectivity for already established connections and to permit setting up new connections to existing conferences. We also described a redundancy-based approach, using forward error correction (FEC), and dispersing the FEC'ed data among disjoint routes. We thus demonstrate the feasibility of building robust protocols for multi-party real-time communication, *without* diluting the strength of the performance guarantees offered, or sacrifing the system performance in the common case, i.e., when all components work correctly.

Acknowledgment

The authors would like to thank Mark Moran for many useful discussions. 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, Tektronix, Digital Equipment Corporation, Hitachi, Ltd., Mitsubishi Electric Research Laboratories, Pacific Bell, 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] Hosame Abu-Amara and Jahnavi Lokre. Election in asynchronous complete networks with intermittent link failures. *IEEE Transactions on Communications*, 1994.

[2] Tony Ballardine, Paul Francis, and Jon Crowcroft. Core Based Trees (CBT): an architecture for scalable inter-domain multicast routing. In *Proceedings of SIGCOMM 93*, San Francisco, CA, September 1993.

[3] Anindo Banerjea. *Fault management for real-time networks.* PhD dissertation, University of California at Berkeley, December 1994.

[4] Anindo Banerjea, Colin Parris, and Domenico Ferrari. Recovering guaranteed performance service connections from single and multiple faults. Technical Report TR-93-066, International Computer Science Institute, Berkeley, California, November 1993.

[5] Riccardo Bettati, Domenico Ferrari, Amit Gupta, Wendy Heffner, Wingwai Howe, Quyen Nguyen, Mark Moran, and Raj Yavatkar. Connection establishment for multi-party real-time communication. In *Proceedings of Fifth International Workshop on Network and Operating Systems Support for Distributed Audio and Video*, Durham, NH, April 1994.

[6] Robert Braden, David Clark, and Scott Shenker. Integrated services in the internet architecture: an overview. Request for Comments (Informational) RFC 1633, Internet Engineering Task Force, June 1994.

[7] David Clark. The design philosophy of the DARPA internet protocols. In *Proceedings of ACM SIGCOMM'88*, pages 106–114, Stanford, CA, August 1988.

[8] 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.

[9] Stephan E. Deering. *Multicast routing in a datagram internetwork.* PhD dissertation, Stanford University, December 1991.

[10] L. Delgrossi and L. Berger. Internet stream protocol version 2 (ST2) protocol specification - version ST2+. Request for Comments (Standard) RFC 1819, Internet Engineering Task Force, August 1995.

[11] Luca Delgrossi, Christian Halstrick, Ralf Guido Herrtwich, and Heinrich Stüttgen. HeiTP: a transport protocol for ST-II. In *Proceedings of GLOBECOMM*, pages 1369–1373 (40.02), Orlando, Florida, December 1992. IEEE.

[12] Luca Delgrossi, Ralf Guido Herrtwich, and Frank Oliver Hoffmann. An implementation of ST-II for the Heidelberg transport system. *Journal of Internetworking Research and Experience*, September 1993.

[13] Luca Delgrossi, Ralf Guido Herrtwich, Frank Oliver Hoffmann, and Sibylle Schaller. Receiver-initiated communication with ST-II. preliminary version, September 1993.

[14] Luca Delgrossi, Ralf Guido Herrtwich, Carsten Vogt, and Lars C. Wolf. Reservation protocols for internetworks: a comparison of ST-II and RSVP – extended abstract. In *Fourth International Workshop on Network and Operating System Support for Digital Audio and Video*, 1993.

[15] Luca Delgrossi, Ralf Guido Herrtwich, Carsten Vogt, and Lars C. Wolf. Reservation protocols for internetworks: A comparison of ST-II and RSVP. In *Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 199–207, Lancaster, U.K., November 1993. Lancaster University.

[16] Robert Doverspike. A multi-layered model for survivability in inter-LATA transport networks. In *Proceedings of GLOBECOMM*, Phoenix, AZ, December 1991. IEEE.

[17] Domenico Ferrari, Anindo Banerjea, and Hui Zhang. Network support for multimedia: a discussion of the Tenet approach. *Computer Networks and ISDN Systems*, pages 1267–1280, July 1994.

[18] Domenico Ferrari and Amit Gupta. Resource partitioning in real-time communication. In *Proceedings of IEEE Symposium on Global Data Networking*, pages 128–135, Cairo, Egypt, December 1993.

[19] Domenico Ferrari, Amit Gupta, and Giorgio Ventre. Distributed advance reservation of real-time connections. In *Proceedings of Fifth International Workshop on Network and Operating Systems Support for Distributed Audio and Video*, Durham, NH, April 1995.

[20] Amit Gupta and Domenico Ferrari. Resource partitioning for multi-party real-time communication. Technical Report TR-94-061, International Computer Science Institute, Berkeley, California, November 1994. Also to appear in IEEE/ACM Transactions on Networking, October 1995.

[21] Amit Gupta and Domenico Ferrari. Admission control for advance-reserved real-time connections. In *Proceedings of IEEE HPCS 95*, Mystic, CT, August 1995.

[22] 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.

[23] Amit Gupta and Mark Moran. Channel groups: A unifying abstraction for specifying inter-stream relationships. Technical Report TR-93-015, International Computer Science Institute, Berkeley, California, March 1993.

[24] Ralf Guido Herrtwich and Luca Delgrossi. Beyond ST-II: fulfilling the requirements of multimedia communication. In *Third International Workshop on network and operating system support for digital audio and video*, pages 23–29, San Diego, California, November 1992. IEEE Computer and Communications Societies.

[25] Sugih Jamin, Peter Dantzig, Scott Shenker, and Lixia Zhang. A measurement-based admission control algorithm for integrated services packet networks. In *Proceedings of SIGCOMM 95*, Cambridge, MA, August 1995.

[26] A. Lazar and C. Pacifici. Control of resources in broadband networks with quality of service guarantees. *IEEE Communication Magazine*, pages 66–73, October 1991.

[27] K. Marzullo and F. Schmuck. Supplying high availability with a standard network file system. In *Proceedings of DCS*, pages 447–455, San Jose, Calif., June 1988.

[28] Danny Mitzel, Deborah Estrin, Scott Shenker, and Lixia Zhang. An architectural comparison of ST-II and RSVP. In *Proceedings of INFOCOM 94*, Toronto, CANADA, June 1994.

[29] Danny Mitzel and Scott Shenker. Asymptotic resource consumption in multicast reservation styles. In *Proceedings of SIGCOMM 94*, London, UK, September 1994.

[30] Mark Moran and Riccardo Gusella. System support for efficient dynamically-configurable multi-party interactive multimedia applications. In *Proceedings of Third International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 143–156, San Diego, CA, November 1992.

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

[32] Colin Paris, Hui Zhang, and Domenico Ferrari. A mechanism for dynamic reroute of real-time channels. Technical Report TR-92-053, International Computer Science Institute, Berkeley, California, April 1992.

[33] Colin Parris. *Dynamic connection management for real-time networks.* PhD dissertation, University of California at Berkeley, August 1994.

[34] Colin Parris and Anindo Banerjea. An investigation into fault recovery in guaranteed performance service connections. Technical Report TR-93-054, International Computer Science Institute, Berkeley, California, October 1993.

[35] Colin Parris, Hui Zhang, and Domenico Ferrari. Dynamic management of guaranteed performance multimedia connections, April 1993. to appear in ACM Journal of Multimedia Systems.

[36] 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.

[37] 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.

[38] K. V. S. Ramarao. Commitment in a partitioned distributed database. In H. Boral and P.-A. Larson, editors, *Proceedings of SIGMOD 1988*, pages 371–378, Chicago, Illinois, June 1988. ACM Press.

[39] R. Schlichting and F. Schneider. Fail stop processors: An approach to designing fault-tolerant computing systems. *ACM Transactions on Computer Systems*, 1(3):222–238, 1983.

[40] Claudio Topolcic. Experimental internet stream protocol, version 2 (ST-II), October 1990. RFC 1190.

[41] Yih-Kuen Tsay and Rajive L. Bagrodia. Fault-tolerant algorithms for fair interprocess synchronization. *IEEE Transactions on Parallel and Distributed Systems*, 5(6), June 1994.

[42] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. RSVP: A new resource reservation protocol. *IEEE Networks Magazine*, 31(9):8–18, September 1993.