The Dynamic Management of Guaranteed Performance Connections in Packet Switched Integrated Service Networks

Colin Parris and Domenico Ferrari
The Tenet Group
Computer Science Division, University of California, Berkeley
and International Computer Science Institute
UCB Technical Report CSD-94-859

Abstract

The Dynamic Management of Guaranteed-Performance Connections in Packet Switched Integrated-Services Networks

Colin James Parris

Doctor of Philosophy in Engineering - Electrical Engineering and Computer Sciences

University of California at Berkeley

Professor Domenico Ferrari, Chair

The communication infrastructure of the future must provide efficient support for applications with diverse traffic characteristics and performance requirements. Currently these applications are supported using several specialized networks that accommodate the different services (e.g., cable networks for video, phone networks for voice, and so on); however, technological advancements in the fields of microelectronics and optics have made it possible to *integrate* these services on a single network. These *Integrated Services Networks* support a wide range of qualities of services to the client and provide many advantages, which include large economies of scale, increased network management capabilities, improved statistical multiplexing, and ubiquity of access. The qualities of service offered by these networks include *guarantees* on various performance indices for a client's specified traffic characteristics; these services are often referred to as *Guaranteed Performance Communication* (GPC) services.

These GPC services provide performance guarantees in terms of throughput, delay, delay jitter and loss rates, and adopt a connection-oriented, fixed-routing, reservation-oriented approach to achieve these guarantees. In such an approach, resource allocation and route selection decisions are made before the start of the communication on the basis of resource availability and real-time network load at that time,

and are usually kept for the duration of the communication. This rather *static* resource management approach has certain limitations: it does not take into account (a) the dynamics of the communicating clients; (b) the dynamics of the network state; and (c) the tradeoff between quality of service and network availability, thus limiting the *flexibility* or *adaptability* of these guaranteed-performance services. In order to accommodate the dynamics of client demands and network state, it is necessary that the GPC services be *flexible*.

In this thesis, we examine this problem of *flexibility* of GPC services in wide-area packet-switched networks, and present a solution by *proof of concept*; that is, we designed a dynamic resource management scheme, analyzed its behavior through simulation experiments, and implemented a prototype of the scheme. This dynamic resource management scheme, called the *Dynamic Connection Management (DCM)* scheme, provides the network with the capability to dynamically modify the traffic characteristics, the performance requirements, and the routes of any existing guaranteed-performance connection. We begin this examination by providing several examples of the dynamics of the client demands and of the network state to motivate our work, and we continue with a review and critique of various proposed solutions. We then present the concept of Dynamic Connection Management and discuss its components: namely, the *DCM scheme* and the *DCM Policies*. The *DCM scheme* is a collection of algorithms and mechanisms that permit the runtime modification of the traffic and performance parameters, and the route of a guaranteed-performance connection. The DCM scheme is guided by high-level management policies, called the *DCM policies*, that determine when modification is permissible in the network and the values of the appropriate parameters to be modified. The focus of this thesis is the *DCM scheme*.

The DCM scheme is an enhancement of the *Tenet* GPC service; it is based on three algorithms: the DCM channel administration algorithm, the DCM transition algorithm, and the DCM routing algorithm; and it is subject to the DCM modification contract. This contract specifies the degree of disruption that a client may experience during a modification. This degree can range from no disruption to a bounded number of performance violations. The channel administration algorithm conducts the admission control tests and reserves the appropriate network resources to ensure that the performance guarantees of the modified channel are satisfied during and after modification. The DCM transition algorithm ensures that the performance violations specified in the DCM modification contract are adhered to during modification. The DCM routing algorithm determines a route from the source to the destination host according to the traffic and performance requirements and other administrative factors.

The DCM scheme also supports mechanisms that enable modifications to a connection to be made to a segment of the connection (*local control*) or to the entire connection (*global control*). Furthermore, faster establishment and modification is also possible as the DCM scheme uses the *intelligent restart* establishment mechanism, which utilizes the real-time network and client state to compute the path before establishment and the *time value* of the network state information to bypass unavailable links during establishment. The DCM scheme was verified and analyzed by a series of simulation experiments. These simulation experiments indicated that the scheme is functionally correct and that the performance of the scheme is very acceptable given our time scales of interest.

In completing the *proof of concept*, we implemented the DCM scheme, and conducted several initial measurement experiments on this prototype implementation. The implementation provided the basic management mechanisms, using a standardized "open" management framework, namely the Simple Network Management Protocol version 1 (SNMPv1), by which guaranteed-performance connections can be monitored and controlled. In our implementation the data delivery protocols used were the transport-level message protocol and network-level protocol of the *Tenet Real-Time Protocol Suite*; namely, the *Real-Time Message Transport Protocol (RMTP)* and, the *Real-Time Internet Protocol (RTIP)*. Management Information Bases (MIBs) were designed for these data delivery protocols and the DCM scheme, and are used provide the monitoring and control capabilities. Results from the initial monitoring and control experiments, which were conducted on a local-area testbed, indicated that the prototype is effective in supporting the monitoring and control of guaranteed-performance connections.

Committee Chair: ____________________________

Professor Domenico Ferrari

Chapter 1

Introduction

1.1 Perspective

Technological advances within the last decade have fostered a period of unprecedented growth in computer and communications systems. The main contributions to these highly touted technological advances were made in the fields of fiber optics and micro-electronics. These advances are responsible for the increase in speeds and capacities of many of the components used in computer and communication systems (i.e. transmission media, switches, memory, and processors) [23], for the reduction in size and cost of these components, and for their increased reliability. The growth that has been introduced by these advances has not only increased the functionality and reliability of these systems, but also made their presence wide-spread by permitting the production of systems that are more useful and affordable.

Coupled with this technological evolution is a major paradigm shift in the needs of the users. This new paradigm is the desire of users to *communicate*. This need for communication is depicted in user demands for total information access, universal connectivity, and interactive communication. Associated with this new paradigm is the use of multiple media types (video, audio, data) for the presentation of information or during interactive communication. The significant increase in the number of video and audio conferencing, data navigation, scientific visualization, and medical imaging applications are indicators of this shift.

Currently these communication needs are satisfied using several specialized networks which accommodate the different services needed for each media type. The cable TV networks are designed to support the broadcast of analog video signals and require high rate, fixed bandwidth, low jitter, simplex services. The telephone networks are used to support interactive voice communication, and require a low rate, fixed bandwidth, low-delay, low-jitter, full duplex service. Data communication is conducted over data networks. These data networks are designed to support communication between computers and do not provide any guarantees in their service. While the current video and audio networks provide guaranteed performance services, the data network provide a "best-effort" service (i.e., they do not provide guarantees on throughput, delay, or jitter).

To support these user needs while taking advantage of the technological advantages, there has been a concentrated effort to integrate the multiple services required so that they can be provided within a single network. A successful integration will provide many advantages, including large economies of scale, improved statistical multiplexing, increased network management capabilities, and ubiquity of access. These new networks will support applications with diverse traffic and performance specifications ranging from the extremely stringent (i.e. high throughput, low delay, low jitter, low loss rate) to best effort ones.

The key motivation behind these *integrated services networks* is to provide a wide range of qualities of service to a client and to utilize statistical multiplexing to increase resource utilization. The qualities of service offered must include guarantees on various performance indices for a client's specified traffic characteristics. Inherently, this motivation seeks to increase the *adaptability* or *flexibility* of the network. Thus, these new networks will not only be capable of adapting to the quality-of-service (QOS) needs of current applications but to the wide variety of QOS needs of future applications. Statistical multiplexing will ensure that this flexibility is supported while efficiently utilizing the networks resources. *Flexibility* can thus be thought of as the ability of the network to adapt itself efficiently to the support of many different quality-of-service requirements.

Current packet-switched networks are somewhat flexible as they can support applications with different traffic characteristics; however, they cannot offer performance guarantees. Circuit-switched networks can offer performance guarantees; however, they are inflexible as they can only offer these guarantees over a very limited menu of traffic characteristics (i.e., usually one or two classes of traffic). An integrated-services network will provide *flexible* services in that it must provide performance guarantees over a wide range of values and combinations of traffic characteristics.

There are, however, several levels of flexibility. The initial concern of efficient support for multiple guaranteed qualities of service has been addressed by several approaches [22, 51, 59, 3, 13, 40, 33, 14, 60] which can be collectively called *Guaranteed Performance Communication* (GPC) Schemes. A review of many of these schemes is provided in Chapter 2. These schemes are intended to make the networks adaptable to a client's initial quality-of-service requirements; however, in many cases a client's quality-of-service requirements will change over the duration of the client session. Also, as the network state changes, it may be possible to satisfy a client's quality-of-service needs that could not initially be satisfied. In other cases, a change in the network state (due to client requests at various times for various qualities of service, link and node faults, mobile host movement, and so on) could result in the need for the network to adapt itself so that its state moves to a better position in the state space[1]. These network state changes may necessitate changes in the routes of various active quality-of-service sessions or the modification of these sessions. These adaptive changes modify the network services themselves to accommodate the needs of the client demands and of the network. The services provided by the GPC schemes are achieved by managing the resources in the network. In order to introduce this new level of flexibility, it is now necessary to do *adaptive* or *flexible* resource management. This thesis studies the problem of adaptive resource management in the context of guaranteed performance communication services.

1.2 Network Environment

In our network model we assume that clients communicate through an internetwork of arbitrary topology, which may consist of several local area and/or wide area packet-switched networks. These networks can support fixed- or variable-sized packets. While fixed size packets technology has been chosen (in the form of Asynchronous Transfer Mode (ATM)) as the transfer mode for future Broadband Integrated Service Networks (BISDN) [12], there are still several advocates of variable-sized packet technology in the computer communication community [13, 15]. It is our belief that these two technologies will co-exist in the future, with variable-size packets used at the internetworking level and fixed-size packets at the network level. In most of our examples we depict local hosts connected to LANs, which are in turn connected by WANs. In each LAN the gateways or routers act as switches, whereas in the WANs switching may be accomplished with a dedicated packet switch. In this thesis the term *switch* will be used interchangeably to refer to gateways, routers, or dedicated switches.

In this work we assume that a) switches are store-and-forward and non-blocking, b) queuing occurs at the output ports of a switch, and c) there is no interference between packets arriving on different input links which are destined for different output links. Also, by *non-blocking* we mean that packets arriving at an input link can be routed directly to the appropriate output link without conflicts in the switching fabric.

With these assumptions a communication session between users (*a.k.a. a connection*) can be modeled as traversing a number of queuing servers, where these servers model the output link of a switch. In our WAN model the output link model is that of the output link of an ATM switch [2, 30], whereas in our LAN model both the output link and the gateway or router processor may have to be modeled. In this latter case, both the processor and the output link are modeled as queuing servers in tandem, with the connection traversing both servers at each switch. In our model, the link delay (i.e., the propagation

[1]This position may be optimal or nearly optimal in terms of availability to client demands, network utilization, or performance criteria.

delay in the case of a physical link or the total cumulative delay incurred in the lower level subnetworks in the case of logical link [19]) of each packet has a known and finite bound. It should be mentioned that this assumption may not be satisfied by some links governed by contention-based protocols (e.g., Ethernets); however, it can be satisfied on other types of LANs such as token rings (e.g., FDDI rings).

Clients communicate through the network by requesting services from the network. In the integrated-services networks that we consider in this work, services can be classified as guaranteed performance services or best effort services. In guaranteed performance services (a.k.a. *real-time services*), a client requests a *connection* by specifying its traffic characteristics and performance requirements to the network, the network determines if these requirements can be satisfied by applying a series of admissions (i.e., resource reservation) tests, and *accepts* or *denies* the client's request based on the result of these tests. The acceptance of a guaranteed performance connection can be thought of as providing a contract between the client and the network, where the network guarantees the specified performance requirement provided that the client obeys its traffic specification. With best effort service, the level of performance received by the client at each instant in time is totally dependent on the network state at that instant (i.e., there are no a priori performance guarantees given to the client). Currently this type of service is that experienced in all conventional data networks.

The initial research on GPC schemes has provided us with some desirable properties that should be present in these schemes. At this point we will describe these properties and refer the reader to Chapter 2 for a more detailed explanation of the schemes. These properties, as detailed in [21], are as follows: 1) the interface offered by the network should be general and parameterized, 2) the scheme should be applicable to a wide variety of internetworking environments, and 3) the performance requirements of the clients should be guaranteed a priori.

A useful interface should not confine its service offerings to a limited menu of possibilities (e.g., HDTV, CD-quality sound, and so on; or low-delay high-throughput communications, high-delay, low-jitter communications, and so on.); rather, it should support a broad continuum of values, as well as combinations of values, for performance requirements. This does not preclude the offering of menu where useful; rather, it allows the easy extension of the menus, at will, where applicable. As useful application servers may be widely dispersed, GPC schemes should be easily implemented over a broad spectrum of internetworks. Clients who make the choice of guaranteed performance services are concerned with provable performance requirements, as these services will primarily be used to create value-added services (video and audio conferences, video-on-demand services) which they will offer to their users[2]. As these value-added services usually provide some form of contract that guarantees their services to their buyers, the providers in turn must rely on the guarantees provided by the network in order to ensure

[2]These guarantees need not necessarily be rigid or deterministic, but can take the form of statistical performance guarantees.

that the buyer's contract is not violated.

The desirable properties presented above give rise to several features that are usually provided by GPC schemes. The features that are commonplace to most schemes are:

- the schemes provide to the client, a priori, guaranteed traffic specifications and performance requirements[3],
- the schemes are usually connection-oriented and resource reservation based, and
- the schemes do resource allocation and route selection at connection establishment time, and usually fix the resources allocated and the route for the duration of the connection.

All GPC services have a component of their agreements or contracts that describe the client's traffic characteristics, and another component that specifies the performance requirements that the network will guarantee. As the network needs to guarantee these performance requirements, it must be aware of the state of its resources (i.e., its resource load). The network uses this resource state or load in order to determine if it can accept each new connection into the network. Acceptance is contingent on the fact that the new connection must be provided with the required performance while ensuring that the performance requirements of existing connections continue to be met. To determine if a new connection can be accepted, the network must be aware of the resources that have been reserved by existing connections for use during data transfers. The acceptance of any new connection in the network essentially coincides with the reservation of resources for that connections; *admission tests* are used to determine if resources are available for that connection, and to reserve these resources if they are indeed available. In order that a client's traffic and performance requirements are met, the resources that are reserved for each client must be allocated to that client on a node by node basis. This type of allocation suggests the need for efficiently retaining state information inside the network that must be present for the duration of the connection. This efficiency is best achieved by using a connection-oriented mode of communication. As the performance of the connection is closely related to the route used by the connection (i.e., a longer path generally has a larger delay associated with it; the greater the number of hops in a path, the larger the aggregate bandwidth consumed by the connection), the choice of route is important and is usually precomputed to guarantee the performance requirements. As state information is associated with each connection to ensure that it is allocated its due resources, packets belonging to that connection should traverse the route along which this state information is kept to ensure that they receive the appropriate resources as well as to ensure that they do not consume resources that have be allocated to other connections. Hence, guaranteed performance services usually

[3]Some of the schemes are general and parameterized, while others are menu-driven and provide a few "useful" classifications.

require connection-orientedness, resource reservation tests, resource allocation, and pre-computed fixed routes. The decisions of resource allocation and route selection are made based on the network "load" (i.e., on resource reservation levels) at the moment the request is being processed. As the client's needs or the network state change, the resources allocated by the network and the routes do not change; they are usually *static*.

In order to support these performance guarantees, two levels of control are needed. At the first level (the connection level), the resource reservation or *admissions control* tests determine if the client's connection request can be honored and reserve resources for this new connection. The second level of control (the packet level) is the service discipline, which controls the order in which packets are serviced, and ensure that packets traversing the route receive the resources allocated to the connection, thereby meeting their performance requirements. Essentially, the service discipline provides protection for the connection by controlling the interactions of packets from different connections. The service disciplines and the resource reservation tests are coupled, as the service discipline determines some of the resource reservation tests. Many current services disciplines can be used to provide guaranteed performance services subject to the constraints described in [20].

1.3 Problem Specification

The essential features of most current GPC schemes were discussed above; namely, that they are connection-oriented; require fixed routing and resource reservation on a per-connection basis; and make resource allocation and routing decisions at the time of connection establishment, based on the resource availability and real-time network load at that time, and fix them for the duration of the connection. This *static* resource management approach cannot support the inherent dynamics of client requirements and network state, both of which may change during the lifetime of a connection. More importantly, clients' requirements and network state may be interdependent — there is a tradeoff between the quality of service offered to clients and the availability of the real-time service. In order to provide a complete specification of the problem we must define the dynamics of interest in this work. Fig. 1.1 depicts the time scale of some of the more significant network operations.

The service times of a packet are usually on the order of microseconds, the round trip times for packets traversing a connection on the network are on the order of milliseconds, and most connection lifetimes are on the order of minutes to hours. The connections that we are referring to are specifically those created by applications requiring performance guarantees (e.g., video/audio conferencing, scientific visualization, and so on,). The client/network dynamics that are of interest to us operate in the seconds timescale, and are those at the *connection entity level*, that is, they involve changes to the connection as a whole during its lifetime rather that to the packets traversing the connection. These changes are usually

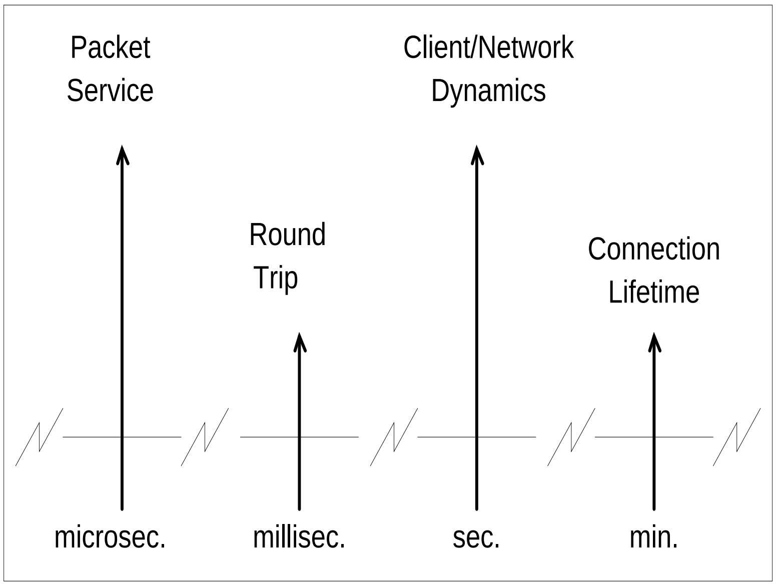

Figure 1.1: Time scale of dynamics

to the characteristics of the connection; namely, the traffic characteristics, the performance parameters, or the route[4]. Several examples of client and network dynamics that are of interest to us are provided below so as to motivate this work. Another category of examples that is presented is that related to the interdependence of these dynamics. This inter-dependence can be seen as a tradeoff between quality of service and network availability: higher quality of service offered to a fraction of the clients may lower the availability of the network, and cause other communication requests to be rejected. Depending on the management policy in effect, it may be desirable to adapt the quality of service offered to the clients based on the load of the network. This adaptation is sometimes referred to as *media scaling*. Of course, in order to stay within the framework of guaranteed performance communication, the adaptation should be graceful (i.e., to be done with minimal or no disruption to the clients) [39]. The proposed resource management algorithms, in their current state, cannot support such graceful adaptations.

In order to support these dynamics in the context of guaranteed performance services, two fundamental capabilities need to be provided to the network, namely, 1) the resource management algorithms must be capable of modifying a connection's current resource reservations so as to accommodate changes in the connection's traffic characteristics, performance requirements, and route while preserving the performance guarantees of all existing connections, and 2) the resource management mechanisms must be capable of supporting these modifications, especially in the presence of a physical change in the route of the connection, while maintaining the contractual guarantees.

The first capability provides the resource management algorithms with the flexibility to adapt to

[4]These changes do not include the traumatic changes of creation and termination of the connection.

parameter (i.e., traffic and performance parameters) and route changes at connection runtime, while the second capability provides the mechanisms that modify the connection's state in the network so as to reflect the new resource reservations. These mechanisms must also accommodate route changes that may cause disruption in the traffic flow of a connection and possible violation of performance guarantees. With route changes, it is possible that packets from a single source may simultaneously traverse two routes during the transition from the original route to the new route, thus presenting the potential for misordered packets.

In order to keep to the first desirable property of a GPC scheme, that is, the property that the interface be general and parameterized, the spectrum over which parameter and route changes can be made should be as broad as possible. However, over the span of this spectrum it may not be possible to provide these modifications while maintaining performance guarantees, especially during the transition from the original connection to the new connection. While there are situations where a client cannot tolerate any violation in the performance guarantees of the connection, there are cases in which some degree of violation is acceptable, specifically where the client itself has requested the modification and is made aware, a priori, of the maximum possible performance violations (i.e., a *bound* is provided on the number of performance violations or the length of the performance violation period) during the transition from the original to the new connection. Therefore, a modification contract is needed to provide contractual guarantees over this transition period. Another issue that needs to be addressed in providing this flexibility is the efficient use of network resources. An increase in efficiency can be realized by choosing a connection's route in the network so that all performance guarantees can be met while attempting to optimize along several network efficiency criteria [5].

The new GPC schemes must provide the network with the algorithms that permit the modification of traffic characteristics, performance requirements, and routes of connections subject to modification contracts, and the mechanisms to control both the connection state updates and the performance disruptions that may be caused by route changes. In this thesis we provide the resource management algorithms and mechanisms, in the context of the *Tenet* guaranteed performance scheme [21], that give a network the ability to adapt to the dynamics of the client demands and of the network state. This new scheme is called the *Dynamic Connection Management* (DCM) scheme, with the new algorithms being collectively referred to as the *DCM Algorithms*.

The next three subsections provide examples to motivate the need for Dynamic Connection Management algorithms. We will divide our examples into three categories: a) dynamics of client requirements, b) dynamics of network state, and c) tradeoff between quality of service and network availability. The final subsection examines some possible alternative approaches to this problem.

[5] These criteria are discussed in Chapter 3.

1.3.1 Dynamics of Client Requirements

There are many situations in which clients may require different qualities of services (i.e., different amounts of resources) during the lifetime of a conversation. For example, in the case of a still image browser, where degradation of quality is not allowed [48], variation of the browsing speed corresponds to the need for different amounts of network bandwidth. If the network does not provide a mechanism to dynamically adjust the bandwidth allocated to the connection, the visualization program will have to reserve resources according to the maximum requirements of the client at the beginning of the connection, which will result in wasted resources. However, if the client is allowed to change the amount of resources reserved for the connection dynamically during the lifetime of the connection, the client would have the option of reserving just enough resources for playback at the beginning of the program, and of acquiring more resources later if browsing at higher speed is needed. This type of application, lossless still image browsing, is very useful in the areas of scientific visualization and medical imaging, where a sequence of related images is browsed in search of minute changes between successive images. These changes can be to the boundary of a liquid or a weather front, where lossy compression cannot be used as it may remove the small changes that are the target of the search.

Allowing the clients to dynamically adjust the traffic or performance parameters will also reduce the burden on the clients of having to set the parameters correctly the first time. As mentioned previously, the interface the network exports to the communication clients must be *general* and *parameterized*; however, as the interface is general, it may be difficult for the clients to estimate the parameters according to the model. This problem is worsened by the fact that, in the current proposed resource management solutions, once the parameters are specified, they remain fixed during the entire lifetime of the connection. This will force the clients to act conservatively, and, most likely, reserve more resources than needed. If there is a mechanism to dynamically change the parameters of a connection, another mechanism can be added to estimate the appropriate parameters for the connection automatically during data transfer.

Multicast connections also provide a rich environment for dynamic management. With multicast connections it is often the case that the addition of a new member to a multicast session already in progress produces a multicast tree that is not minimum-cost. This situation may occur even if the previous multicast tree was minimum-cost and the branch connecting the new member to the tree is also minimum-cost. In these cases, the main branches of the tree may be dynamically and transparently rerouted to produce the minimum-cost tree that includes the new member.

This situation is illustrated in Figure 1.2, where initially source **S** was transmitting to the destination set **D1**, which was comprised of high bandwidth workstations, WS, and a low bandwidth personal computer, PC. Then three new workstations, indicated by ?, attempted to join the multicast session.

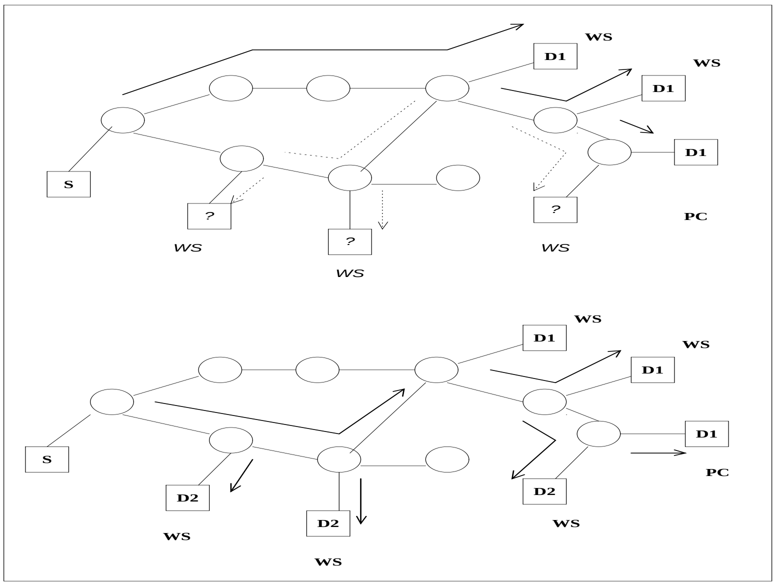

Figure 1.2: Multicast Management

The dotted lines in the topology in the upper portion of the figure indicate the additional links needed to accommodate these new members. If the capability of dynamically modifying connections were present, the new multicast tree (which includes the new destination set **D2**) could be made considerably cheaper by the rerouting of its main trunk, as shown in the lower portion of Figure 1.2.

With the current demand for wireless real-time connections [44], dynamic connection management provides the ideal primitives for wireless network support. If we assume a model in which the portable clients are intelligent [49], the movement of the client among base stations can be thought of as the rerouting of the connections, which can be supported without disruption by DCM[6].

1.3.2 Dynamics of Network State

In addition to the dynamics of client requirements, the network state also changes during the lifetime of a connection. Examples are failures or exceeded error thresholds in links and nodes in the network. In connectionless networks, packets can be dynamically rerouted when there are failures inside the network. We would like to port this flexibility of dynamic rerouting to connection-oriented networks. This flexibility is important in that a connection that traverses a failed link or whose error thresholds have been exceeded across a link can be quickly rerouted (this could be achieved by rerouting that portion of the connection that traverses the offending link) to one or more other links, thereby reducing the number of lost or corrupted packets occurring on this connection. These error thresholds are

[6] We assume here that the underlying data link level technology is such that we can bound the errors introduced by the medium, and that the performance guarantees made are statistical.

quantifications of error tolerances appropriate to each connection; for instance, in video connections, the number of consecutive packets or cells lost may be the error tolerance of choice rather than the total number of packets or cells lost. This flexibility has been demonstrated using DCM, through simulation experiments, described in references [38, 9].

Another example of the use of DCM can be seen in network load balancing, where a less that optimum network state can result as clients' requests for creation and termination of guaranteed performance connections can occur in a non-uniform manner throughout the network. These request can leave the network in a state where it is not efficiently utilized and can result in reduced availability for guaranteed performance clients and high delays for best-effort clients. DCM can alleviate this problem, with the support of efficient load balancing policies, by transparently rerouting guaranteed performance connections, thereby increasing the efficient utilization of the network.

A simple example is shown in Fig. 1.3. In the network illustrated in the upper portion of the figure, two guaranteed performance connections have been established from the source **S** to the destination **D**, one connection along the lower path[7] (a 40 Mbps connection) and one along the upper path (a 50 Mbps connection). In this network state, a new request arrives for a 80 Mbps connection from the source **S** to the destination **D**. Since the residual bandwidth along the upper and lower routes is 60 and 50 Mbps, respectively, while there is enough aggregate bandwidth to support the 80 Mbps connection, this is *fragmented* among the two routes in such a manner that the 80 Mbps request cannot be accepted. Similar problems have been addressed in the context of telephone networks by a technique called trunking [41]. Dynamically rerouting the 40 Mbps connection to the lower path can solve the fragmentation problem in this situation, as it permits the 80 Mbps connection to be established along the upper path. This is shown in the network in the lower portion of the figure.

1.3.3 Quality of Service vs. Network Availability

One of the criticisms against reservation-based algorithms is that they do not address the tradeoff between quality and availability of service. Although the network will guarantee the quality of service to the connections already established, it may have to block other connections due to lack of resources. This is necessary if the qualities of service of the established connections cannot be compromised. However, there are certain applications that have the ability to adapt to different qualities of service, and may be willing to reduce their quality requirements in cases of network saturation. For example, some video coders are designed with tunable parameters so that the compression ratio can be adjusted to output streams with different bit rates [17, 29, 26]. When the network is less loaded, we would

[7]The maximum bandwidth of each link in the network is 100 Mbps. For the sake of simplicity, we assume that only one performance parameter, the throughput, is guaranteed; however, this example can be applied to other performance parameters such as delay or delay jitter.

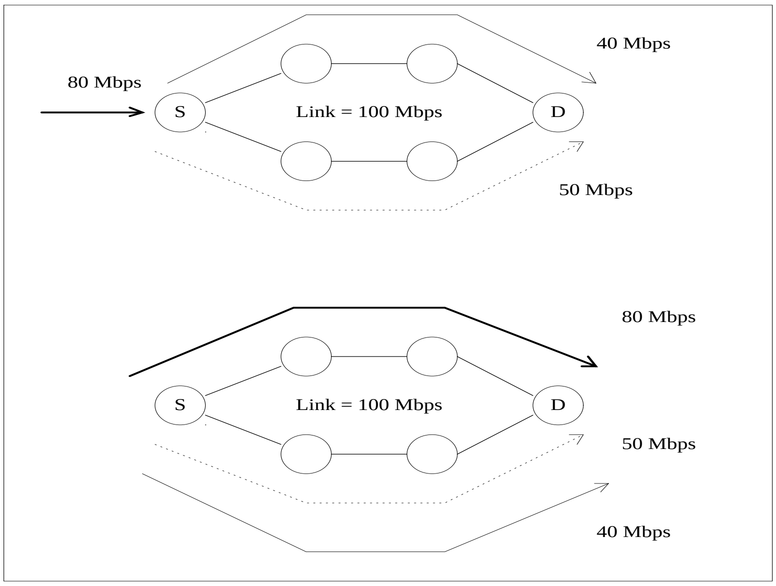

Figure 1.3: Load Balancing Example

like the compression ratio to be small so that we can have higher video quality; however, when the network is close to saturation, we would like to increase the compression ratio so that we can admit more connections. Such adaptation, also known as *media scaling* [18], has been proposed in the context of datagram networks [54, 27], but without providing any guarantees before or after the change. By using Dynamic Connection Management on the guaranteed performance network connections, we can better address the tradeoff between quality of service and availability of service: parameter adjustments will only be applied with the consent of the application [8]. Simulation experiments using the DCM algorithms have also been conducted in a media scaling context; a description of them can be found in reference [39].

1.3.4 Possible Alternative Approaches

Our approach to the problem discussed previously is to introduce mechanisms that allow modification of traffic and performance parameters and of the route of a guaranteed performance connection at the networking layer. There are two other possible alternative approaches.

In the first alternative approach, when there is a desire to change parameters, the end system establishes another connection with the new parameters, switches the traffic from the old connection to

[8]If network service is free, all clients will ask for the highest quality of service. Quality of service only makes sense when there is an adequate pricing structure [37, 16]. We assume that the pricing policy takes into account the traffic and performance parameters as well as the duration of a connection, thus providing incentives for clients to adjust their quality of service on a voluntary basis. Also, contracts where media scaling is permitted will carry a lower price tag than those where scaling is not permitted

the new connection, then tears down the old connection. In such an approach, the network does not know the relationship between the old and new connections, thus cannot share resources between them. Without the sharing of resources network, availability will be adversely affected. Also, mechanisms like rerouting a channel locally[9], which is used by DCM to accelerate the transition time, cannot be used in such an approach.

The second alternative approach is to establish additional connections to transmit the excess data when more bandwidth is needed.This second alternative can easily result in resource fragmentation. In resource fragmentation, numerous channels, with low performance requirements, are scattered throughout the network in such a manner that availability of the network to high performance channels is limited while there is obviously enough aggregate resources to support these channels. This also requires the end-system to support multiple connections for one logical stream and provide synchronization among these connections. Also, it is unclear how to degrade QOS for a connection with such an approach.

1.4 Thesis Organization

In this work, we study the issues and sub-problems that must be addressed in designing and implementing dynamic connection management in a Guaranteed Performance Communication scheme. The dissertation proceeds in the following manner: we first review the literature to determine the strengths and weaknesses of previous schemes; we design the new algorithms that permit flexibility in the *Tenet* GPC service; we analyze these algorithms using simulation experiments; and, finally, we implement and evaluate the algorithms and mechanisms in a GPC environment. Each component of our work is provided in a separate chapter, with a summary and description of future work contained in the conclusion.

Chapter 2 provides a review of the literature and an analysis of relevant work. The review is divided into two sections. The first section examines relevant guaranteed performance communication schemes while the second examines routing under performance constraints.

Chapter 3 presents the details of the Dynamic Connection Management scheme. It begins with an overview of the Tenet scheme and of the service discipline Rate Controlled Static Priority (RCSP), which are the building blocks of the DCM scheme. The chapter proceeds with the basic paradigms under which DCM operates, and then provides an in-depth description of the three algorithms that comprise the scheme.

Chapter 4 analyses the scheme through the use of simulation experiments. The experimental setup is first described, and several useful simulation experiments are conducted to determine the validity and usefulness of the scheme.

[9]Local rerouting can be used to accommodate a change in the delay bound of a connection.

Chapter 5 presents the design and an implementation of the DCM scheme. The scheme has been implemented using the Simple Network Management Protocol version 1 (SNMPv1) as the management protocol, on a DEC platform under Ultrix4.2a. This implementation provides both monitoring and control capabilities to the network in that SNMPv1 Management Information Bases (MIBs) can be used by DCM *Network Managers* to give the state of a connection and to establish or modify a connection. The guaranteed performance connections use the Real-Time Message Transfer Protocol (RMTP) and the Real-Time Internet Protocol (RTIP) as their data transport protocols. Results from a measurement study show that the DCM scheme is effective in modifying the parameters of a GPC connection.

Chapter 2

Related Work

There have been several GPC schemes proposed over the last few years. The guarantees, which can be statistical or deterministic, provided by these schemes span a wide range of performance parameters and utilize a variety of traffic models. In the first subsection we provide an overview of the relevant schemes and we examine their flexibility. In the second subsection we survey the literature on routing with emphasis on routing techniques with performance constraints. In that subsection we review routing techniques from both packet switching and circuit switching environments to determine their suitability for use in our work.

2.1 Guaranteed Performance Communication Schemes

The GPC schemes reviewed below represent the early and current research in guaranteed performance communication and, as such, address a variety of sub problems in this area, thereby resulting in wide variations in their proposed solutions. Some schemes provide connection management (i.e., resource management) solutions [60] , others provide guaranteed data transport solutions [59], and yet others provide comprehensive solutions involving both connection management and data transport services [21]. All widely known GPC schemes are presented regardless of the type of solution; however, it should be noted that the solutions more relevant to this work contain a resource management component and operate at the network layer level[1]. As the DCM scheme is based on the *Tenet real-time* scheme, which is a comprehensive scheme, we defer a review of the *Tenet* scheme to Chapter 3.

The Flow Protocol provides guarantees only for average throughput bounds [59] on a flow (i.e., a connection). It does not provide guarantees for delay, delay jitter or loss bounds. While end-to-end delay bounds are not guaranteed, these delays can be somewhat reduced by increasing the throughput resources reserved; however, the quantitative relationship between the percentage of throughput

[1]The network layer level that we refer to is the third layer from the bottom in the ISO OSI reference model.

resources reserved and the delay reduction has yet to be determined. The Flow Protocol reserves resources and uses rate control to enforce a source's traffic characteristics. The *VirtualClock* scheduling discipline is used to ensure that the bandwidth reserved by a client is accessible to that client and to provide isolation from other clients. There are no features in this protocol for the dynamic modification of flows.

The Session Reservation Protocol (SRP) provides throughput and delay guarantees for a session by reserving resources at each node [3]. The client is required to specify its traffic parameters using the DASH resource model. SRP creates sessions at each node, which represent reservations of a part of the capacity of each resource. These reservations are sufficient to guarantee the local throughput and delay requirements of the client. These sessions are then linked together along a fixed route to form an end-to-end session, which provides the end-to-end performance guarantees. The end-to-end session establishment and philosophy are similar to that of the Tenet scheme[2], but there are a few differences. The differences are that SRP uses a different traffic model and a different admission control policy, and the control and delivery functions are incorporated within the same protocol. Also, SRP aligns itself closely with IP in that it seeks to achieve performance guarantees for IP-based communication without changing the IP protocol and does not provide guaranteed delay jitter bounds, overflow bounds, or any statistical guarantees. SRP does not employ any mechanisms for the dynamic management of sessions.

The Asynchronous Time Sharing (ATS) approach provides a fixed menu of classes-of-services [32]. There are 4 classes of services. Three of the classes, Class I, II and III, transport user traffic, while the fourth transports network management traffic. Class I traffic is characterized by zero percent contention[3] loss and an end-to-end delay distribution with a narrow spread. Class II traffic is characterized by e % contention packet loss and an upper bound, n, on the average number of consecutively lost packets. Its end-to-end delay distribution has a larger spread than the Class I distribution. The end-to-end delay bound of Class I packets is less than that of the Class II packets. Class III traffic is characterized by zero percent end-to-end packet loss. This is achieved by retransmissions. Real-time connections can only obtain performance guarantees corresponding to those of their class. The service discipline of ATS, MAgnet Real-time Scheduling (MARS), has its activity divided into cycles. A cycle is the period during which a fixed number of cells can be transmitted and is divided into four subcycles, each corresponding to a class-of-service and to the allocation of the link to this class-of-service. The duration of each subcycle is determined dynamically by the MARS scheduler according to the traffic load and the mix. While there is an extensive monitoring capability in the ATS approach, to our knowledge there is no ability to manipulate real-time connections.

[2]The Tenet scheme is described in Chapter 3.

[3]Contention packet loss represents packets that are discarded due to buffer overflow and packets whose end-to-end delay is greater than the maximum set limits.

Clark, Shenker, and Zhang propose an Integrated Services Packet Network (ISPN) architecture that will support guaranteed and "predicted" performance services [14]. In their context, guaranteed bounds on throughput and delay can be provided to a client if the client specifies its maximum sending rate. For this rate the network will inform the client of the offered delay bound; if the delay bound is insufficient, the client must request a higher sending rate so that the delay requirement can be satisfied. For predicted service, the client characterizes its traffic and the delay and loss rate it can tolerate. The admission control scheme determines if there are sufficient resources and stability in the network to accommodate the client. Due to network load fluctuations, predicted-service clients experience fluctuations in their performance specifications, to which they can choose to adapt to accommodate these perturbations. The network naturally does its best to ensure that there is high stability in its load. Clark et al. propose a *unified scheduling algorithm* capable of supporting both guaranteed and predicted traffic. The ISPN architecture does not address any of the network management issues, and provides no capability for channel modification.

The Stream Protocol, Version II (ST-II) [51] is an experimental internetwork-layer stream protocol developed for the Internet. ST-II decouples control from data delivery, as it uses the ST-II Control Message Protocol (SCMP) to establish and teardown connections. Connections can be unicast or multicast. Connections are established by sending an SCMP message to each hop along the route, which determines if resources are available for the connection. The ST-II agents in the intermediate and destination nodes utilize the parameter flow specifications[4] to determine the availability of resources. Some of the parameters included in the specification are minimum allowable/deliverable bandwidth, maximum allowable/deliverable delay, delay variance, and so on. The specification of ST-II does not detail any of the resource reservation policies. The design of these policies is left to the implementor. The intermediate nodes modify the flow specification to reflect the available resources they have reserved. The destination can accept or reject a request for a connection by examining the modified flow specification and sending its response back to the source. If the source and destination are in agreement, the connection is established and data transport can begin. The SCMP specifications include a CHANGE message that can be used to modify parameters but no formal descriptions are given of this CHANGE functionality, and no implementations have been reported in the literature. However, it should be noted that there have been oral reports that the ST-II protocol implementation using the HeiRAT resource management algorithms seeks to provide this functionality, but the author is unaware of written confirmation of these reports.

In the Capacity Based Session Reservation Protocol (CBSRP) [50] the user can specify the minimum and maximum values of the desired temporal and spatial resolutions of the media[5] to be transmitted, the

[4]There are 15 flow parameters, with the specification permitting applications to add extra fields.

[5]The temporal and spatial resolutions of the media are the frame rate and the frame size of a stream.

allowable end-to-end delay, and the maximum packet loss rate. The specified values allow the network to assign each client to a particular class of service. When a new client requires the establishment of a session, if the available resources are already saturated, some existing sessions may be forced to reduce their qualities of service (i.e., to modify their parameters), to accommodate the new request. However, this modification of parameters is restricted to the values of the temporal and spatial resolutions of the stream - only bandwidth modifications are allowed, end-to-end delay and loss rate modifications are not supported. The minimum quality of a session ,i.e., the minimum values of the temporal and spatial resolution, is always guaranteed once the session is established. Another limitation of CBSRP is that is has been designed for a local area network environment, while issues associated with a more general internetworking environment have not been addressed.

The Multipoint Congram-oriented High-performance Internet Protocol (MCHIP) [40], is a guaranteed performance service that also provides multicast capabilities and utilizes resource servers to monitor and record channels established and resources available. MCHIP uses as its service primitive a *congram*[6]. MCHIP uses two types of congrams: a user congram (UCon) and a persistent internet congram (PICon). A UCon can be thought of as a soft connection (with no hop-to-hop error control) and can be used to provide strict performance guarantees to applications that need it. PICons are long lived congrams between MCHIP entities, and their purpose is to provide a shared communication channel for a number of applications, thus suitable for carrying datagram traffic. While the services have been designed, to the best of our knowledge the channel establishment procedure and resource reservation policies have not yet been published.

The ReSerVation Protocol (RSVP)[60] is a resource reservation protocol that can be used to reserve resources for multicast connections. The key features of RSVP are: it is receiver-oriented; it allows each receiver in a multicast group to reserve a different amount of resources, to receive different data streams sent to the same multicast groups and to switch between these streams without changing its reservation; also, it supports dynamic memberships and adapts to routing changes by using *soft state*. RSVP does not address real-time data transport and is entirely a vehicle for establishing and maintaining per-connection state information in the switches along the paths traversed by a connection. RSVP keeps soft state information at intermediate switches and leaves the responsibility of maintaining the paths and reserving resources to end users. These end users (i.e., the source and destinations) send refresh messages at intervals (these intervals are maintained by using refresh timers) to refresh the path and the reservations. This use of soft state does permit the modification of traffic and performance parameters and routes; however, it introduces some other problems that the RSVP designers are now attempting to address. As path and reservation messages are sent periodically, there is the possibility of corrupted,

[6]The congram service primitive seeks to combine the strengths of the *con*nection and data*gram* approaches.

lost, and late[7] messages that may result in the removal of state information at an intermediate switch. This removal of state information can cause wide variations in the performance of a connection, as its resources may have already been assigned to another connection. Also, as the routing mechanism and the reservation protocol are decoupled, it is possible that the underlying route may change to a route which does not have adequate resources to support the connection at its current level of performance, thus resulting in a decreased level of connection performance until the connection has been re-established on a suitable route.

The plaNET network [13] is a guaranteed performance scheme that provides guarantees on the bandwidth and loss rate of a connection. It contains both resource reservation and data transfer components in its solution. The scheme is implemented in hardware and uses the notion of *equivalent bandwidth* to determine the resources reserved for a connection. In plaNET an estimator is used, for each connection, to estimate the actual traffic characteristics of the source. This estimate is then used to renegotiate the connection's bandwidth parameters. Thus, it is possible to modify the bandwidth of a connection. However, a drawback of this scheme is that delay bounds and jitter bounds are not supported and we believe that these performance parameters are needed to provide useful real-time services. It should be mentioned that this deficiency is currently being addressed, as delay bounds and jitter bounds are now being incorporated in the plaNET performance parameter set.

2.2 Routing Techniques

Routing on the basis of traffic and performance constraints (a.k.a, *real-time routing*) has been significantly researched in the context of circuit switching environments (i.e., in a telephony context) but only marginally so in packet switching environments. In this subsection we review routing techniques in both environments and discuss their suitability for use in our work.

To maximize the utilization of high-speed GPC networks and reduce the risk of saturation in these networks, real-time routing is very important. As performance guarantees are made along a fixed route, *virtual circuit* routing techniques are most relevant to our work. As we wish to maximize the "run-time" efficiency and reduce the saturation of the network, we consider only dynamic or adaptive routing schemes. Dynamic routing is a network routing technique that routes client channels on routes determined by using the current real-time state of the network, thereby allowing the network to respond quickly and correctly to changes in network loading and facilitating a high utilization of the network's resources. We now present a survey of many of the important routing techniques used in these networks, and provide an analysis of their usefulness in our real-time environment.

[7] This situation may arise due to the difficulty in setting the refresh timers correctly.

2.2.1 Circuit Switched Routing Techniques

Dynamic routing techniques used in circuit-switched networks can usually be divided into two major categories: *time-dependent* and *state-dependent*. In time-dependent routing, preplanned routing patterns, which were computed offline, are entered at fixed times during the day to allocate network capacity for previously forecasted traffic demands. In state-dependent routing, the routing patterns are automatically varied according to instantaneous traffic demands and network status information to respond to traffic variations. Network status information includes such information as the number of successful calls and the occupancies of the trunk groups. The analysis of the information and the selection of routes can be done in a distributed or centralized manner. Usually the collection of load information and the route selection are done periodically, with the selected routing patterns valid for the entire period. Circuit switch routing techniques are almost always applied exclusively in telecommunications, where the backbone network[8] is fully connected and routes are considered as direct or alternate. Direct routes are one-link routes; alternate routes are usually limited to two-link routes. Performance studies have shown that alternate routes with more than two links reduce call acceptance rates. Also, with these networks there is associated a trunk reservation scheme that limits the effect of overflow traffic (i.e., calls that utilize the two-link alternate routes) on calls that require their direct routes. The network management facilities also include Automatic Call Blocking (ACB), whereby the network limits access to certain portions of itself in order to reduce the degradation of service to current users[9]. The main circuit-switching routing techniques in use are summarized below.

Aggregated Least Busy Alternative (ALBA) is a distributed, state-dependent, dynamic routing technique for fully-connected networks [35] . We assume that the network has N nodes and a link with C circuits between every node pair; routes are restricted to have at most two links. The direct link is the single link route between the source and the destination. The alternate route is any of the possible $N-2$ two-link routes between the source and the destination. In ALBA, upon arrival of a call, local information on the state of the links of all possible routes is used to determine the route of a call. In ALBA(k) the (C+1) states of each link, which represent the number of occupied circuits on that link, are lumped into k aggregates $(A_0, A_1, A_2, ..., A_{k-1})$ in order of increasing occupancy level. The largest aggregate A_{k-1} is the set of states with r or less idle circuits; we say that A_{k-1} comprises the set of reserved states and r is the trunk reservation parameter. Routing occurs in the following manner: the arriving call is attempted on the direct route; if a circuit is available, the call is carried along this route; if no circuit is available, the call is attempted on an aggregated-least-busy-alternative two link route. An ALBA two link alternate route for a source/destination pair s, d is one that minimizes $max(A_{si}, A_{id})$, where i is the

[8]This is the lowest level network in the hierarchy of networks that comprise a major telecommunications system.

[9]These situations are called *focused overload situations*. An example of this situation occurred during the last California earthquake, where many people tried to call California at the same time.

intermediate node and A_{si} and A_{id} are the aggregate states of links (s,i) and (i,d) respectively. Any of the $N-2$ alternate routes can be chosen, with ties broken randomly. The arriving call is accepted on a two link alternate route if it does not leave either link in aggregate A_{k-1}, else the call is blocked and lost.

State- and Time-Dependent Routing (STR) was developed by Nippon Telephone and Telegraph (NTT) and combines a learning automaton (the state-dependent portion) and a time-varying method (the time-dependent portion). In STR each node has a route selection list that is updated periodically by a central processor. These route selections are determined by that processor based on the network topology and link size, and the predicted traffic load between these nodes. When an incoming call arrives, it is routed using the route selection list. The call is first offered to the direct route and overflows to an alternate route, specified in the route selection list, if it is blocked. The alternate route is chosen using a learning automaton routing method. In this method a call overflowing its direct route is offered an alternate route. If the alternate route is not congested (i.e., if the number of idle trunks on either of the links does not exceed a set threshold), the call is given the route, and any subsequent call attempts the same alternate route. If the alternate route is congested, another alternate route is attempted by the call. Congestion status on the second link is relayed by using the call-completion signal or the trunk release signal sent back to the source node. This isolated scheme has several variations depending on whether the alternative routing is of the single-overflow type or of the multiple-overflow type. In the single-overflow type only one alternate route is considered if a direct route is not available, whereas in the multiple-overflow type a specified number of alternate routes are considered if the direct route is unavailable. The first available alternate route is chosen for the call. The value of the congestion threshold can be varied to produce other scheme variations. In STR, usually the single-overflow scheme is used with a set non-zero value for the congestion threshold. Performance evaluations for combinations of these variants are given in [34].

Dynamic Non-Hierarchical Routing (DNHR) [5] was developed by AT&T and is a centralized, hybrid time-dependent and state-dependent route selection technique, where the time-dependent factor is determined by forecasted traffic patterns and the state-dependent factor is a response to network load variations. In DNHR, the direct primary route and two-link alternate routes are used to carry traffic between source and destination DNHR tandem switches[10]. The time-dependent routing capability allows prespecified routing patterns to change as frequently as every hour in response to forecasted traffic patterns. In DNHR a call is first offered to the direct route; if no circuits on this route are available, the call is offered to the alternate route set. If blocking occurs on the second link of an alternate two link route, a control message is sent back to the source node so that this blocked call can be routed on

[10]These switches form the highest-level of the telecommunications network hierarchy and are usually intercity exchanges.

another route in the alternate route set (this technique is commonly referred to as *crankback*). These alternate routes (there is a maximum of 14 for each source/destination pair) are examined sequentially until the call is accepted. If the call cannot be accepted by any of the alternate paths, then the call fails. DNHR also includes a state-dependent routing ability, courtesy the NEtwork Management Operations System (NEMOS), which searches for idle trunk capacity on an individual call basis using the trunk reservation parameters as idle thresholds. If idle links are found, they can be incorporated into the alternate route set to increase the offering to an arriving call. This dynamic routing is only used in the case of network link failures and other unusual scenarios.

DNHR has also been extended to provide state-dependent routing based on trunk status information. This new routing technique is called Trunk Status Map Routing (TSMR) [4]. The TSMR concept involves having an update of the number of idle trunks in each of the DNHR trunk groups sent to the centralized network database every T seconds. The database determines a new ordered routing sequence based on the number of idle trunks and returns this sequence to each switch. The new ordered sequence is used for the next T seconds until the next update.

Dynamically Controlled Routing (DCR) [11] is a state-dependent routing method, developed by Bell-Northern Research, which uses centralized routing to determine the best routes, depending on the occupancy of the trunk groups. Each call is first offered the primary route. If this call is blocked, it is then offered to an alternate two-link route. The alternate route is selected on a probability basis and the number of call attempts on alternate routes can be set at some threshold value. A central processor computes route probabilities at fixed intervals based on the residual capacities of the links. These probabilities are usually computed every 10 seconds.

The System for Test Adaptive Routing (STAR) [25] is a state-dependent routing technique developed by the Centre National d'Etudes des Telecommunications (CNET) for dynamic routing in the French Telecom network. In this method, trunk group occupancy information is periodically collected at a central processor. The route selection sequences are updated at each local switch in order of decreasing route residual capacity. The residual capacity of a link is defined as the minimum residual capacity of all trunk groups belonging to the route. Incoming calls are offered first to the direct path and, if blocked, are offered to the alternate paths in sequential order. Crankback may be used to detect blocked calls and do multiple alternate route attempts. Initially the STAR prototypes were implemented with the update period varying from 1 to 2 minutes due to technical constraints; however, the production switches are expected to have an update period on the order of 10 seconds.

Real-Time Network Routing (RTNR) is a decentralized routing technique developed by AT&T to provide routing for future dynamic class-of-service networks, which provide connections for voice, data, and wideband services on a shared transport network [6]. With RTNR, the source switch attempts to route an incoming call onto a direct trunk. If this direct trunk is not available, the switch attempts to find an available two-link route by first querying the destination switch[11] for the busy-idle status of all trunk groups connected to it. The source switch then compares its own trunk group busy-idle status information to that obtained from the destination switch in order to obtain the least loaded two-link path over which the call can be routed. This least loaded two-link path is obtained by comparing load threshold bit maps of the trunks corresponding to the source and destination switches. To obtain a specific class-of-service route, these load threshold bitmaps can be overlayed by an allowed-via-switch bit map which indicates routes that possess this class of service. Of these allowed links the least loaded two-link path is selected. It should be noted that the class-of-service routing provided by RTNR usually refers to bandwidth-based services.

Dynamic Alternative Routing (DAR) is a decentralized (isolated) learning-automaton routing method developed by British Telecom [46]. In a DAR network an incoming call is offered the direct route. If this route is blocked, the call overflows to the currently selected two-link alternate route. If the call is also blocked at this alternate route, the call is refused, and a new two-link alternate route is selected at random from possible two-link routes for subsequent calls.

In the previous sections we have reviewed some of the more prominent circuit-switched routing techniques. The major disadvantages of these techniques are as follows: usually only one type of service guarantees is considered (i.e., guarantees are made implicitly by the use of dedicated circuits and by the performance-oriented design of the telecommunication system[12]) and the value of the bandwidth and delay parameter in these circuits cannot be changed; fully connected networks are assumed; in some cases the network traffic can be predicted (i.e., in time dependent routing techniques), and this prediction is utilized by the routing technique. Another drawback of some of the schemes is that centralized control is used.

These disadvantages reduce the usefulness of these techniques. In our environment many different classes of services are to be provided, hence a routing technique that optimizes routing for a single class of service is not useful. Also, the routing techniques surveyed cannot determine routes that provide a broad range of delay and jitter guarantees. The assumption of a fully connected network is not appropriate for the types of network topologies we deal with. These routing techniques are limited to two-link routes. As the topologies that will be encountered are most likely to be relatively sparse, optimizing for two-link routes is not appropriate. The traffic loads that will be seen on these networks will be very dynamic and highly unpredictable. The unpredictability is due to the lack of a convincing model for the multimedia traffic that will be present on these networks. The rate of growth of multimedia

[11] This query is done over the Common Channel Signaling (CCS) Network .

[12] The bandwidth given by a circuit is dedicated to the client for the duration of the call and the delay experienced in a phone call is bounded.

applications and the possible combinations of applications will provide an environment that will render historical call data obsolete; prediction will be almost impossible. The disadvantages present in circuit switching routing techniques do not allow us to utilize these techniques directly.

2.2.2 Packet Switched Virtual Circuit Routing

Although there are many virtual circuit routing techniques, three of them that are particularly suitable for real-time routing. They are the *plaNET* routing algorithm, the scheme used by *Codex*, and a multicast routing scheme proposed by *Kompella et al.*

The plaNET network is a high-speed packet switching network designed to support multiple classes of service [1]. The objective of the source routing technique used in plaNET is to minimize call blocking while providing low end-to-end delay. In plaNET only bandwidth is guaranteed by reservation, but delay is minimized as much as possible. The routing algorithm assumes that queuing delay is not a major issue in high-speed networks, as switches will be very fast and hence the focus of the routing algorithm is to minimize call blocking. Minimization of call blocking is achieved by favoring the shortest path between source and destination, and by load balancing. By favoring the shortest path, less links are used; hence the call blocking effect of this call on other calls in the network is reduced. By load balancing among shortest paths, the load is distributed evenly among the links, and, when this is correctly done, this also tends to reduce call blocking. Results are provided in [1] to verify that these objectives are achieved. The routing algorithm used is a modified shortest path algorithm where the link weights are an increasing function of the link load (this promotes load balancing). This algorithm is further constrained to obtain the minimum number of hops between the source and the destination, thereby reducing the effect of this channel on other channels in the network. Link weights take into consideration the bandwidth of the channel requested and the current load on the link; they are used to make saturated links unavailable and to discriminate among paths with equal number of hops. Among the minimum-hop paths, the path with the lowest total weight is chosen.

The Codex routing scheme permits the routing of virtual circuits and the rerouting of these virtual circuits in response to link congestion levels [28]. A virtual circuit is established between a source and a destination by sending a route message to the destination node, which then uses a source[13] routing shortest-path technique to establish a path from the destination to the source. Thus, the destination node is responsible for routing and rerouting the virtual circuit. This routing algorithm is based on the total cost of all of the links in the network. A link's cost is a function of the excess capacity of the link, the data rate of the connection, the propagation delay, the total weighted traffic, and the priority of the connection. Therefore, the total network cost will be lower if connections with high data rates or

[13]In this case the source responsible for the routing is actually the destination node.

high priorities are routed along routes containing only a few hops. Routes for new paths are selected to minimize the increase in network cost. This is accomplished by assigning to each link the increment in the link's cost if the new connection were accepted, and then running a shortest path algorithm to determine the path with the least total cost. This path is the selected route. Rerouting decisions are made by scanning the routes terminating at a node. For each route terminating at a node, computations are made to determine what the traffic characteristics of the network would be if the current route were deleted and a new shortest route established. Using those characteristics, the cost of the current route and the cost of the shortest route are computed. If the latter is less than the former, a secondary path is established. To protect against the possibility that many reroutings will overload a previously lightly-loaded link, only a fraction of the paths are considered for rerouting at any time, and a maximum amount of rerouted traffic can be established on any link over an interval.

Kompella et al. propose a multicast routing algorithm that determines a least-cost tree, spanning from the source to all of the destinations, subject to the constraint that all source/destination routes have a client-specified delay bound [31]. In their algorithm, they initially construct a closure graph on the set containing the source and all of the destination nodes. This graph is constructed from the cost and delay weights assigned to each edge (the edges represent links and the nodes represent packet-switching nodes) and is a complete graph where each edge connecting two nodes represents the minimum cost path where the delay along the path is less than the client-specified delay. The delay along the path is the sum of the propagation and transmission delays. Prim's spanning tree algorithm is then applied to this graph to determine a minimum-cost spanning tree. Heuristic functions are provided for choosing edges in the spanning tree. This tree is expanded into the edges that represent the constrained cheapest paths. As it is possible that the resulting graph is not a tree because the edges in the closure graph represent paths and expanding them may cause loops, a subsequent operation is performed to remove these loops. The result is a minimum spanning tree where each source/destination path is constrained by a specified delay bound. While this is a multicast algorithm, it can easily be applied to the unicast case, hence it merits analysis here.

The plaNET routing technique contains several useful ideas: for example, routes can be determined that support different classes of service; blocking probability can be decreased by reducing the number of links in the route; and the topology considered is similar to those encountered in our environment. However, the routing technique only takes into consideration the bandwidth resources, and not the delay or buffer resources. In the Codex algorithm, bandwidth resources are taken into consideration, and rerouting is examined; however, the delay and buffer resources are not addressed. The problem of achieving a minimum-cost route subject to delay constraints is exactly that we wish to solve, and Kompella et al. propose a useful, though computationally expensive, solution for a multicast environment. With these disadvantages in the circuit and packet switching routing techniques, it was necessary to extract the essence of the advantages from these techniques and use them to develop a suitable set of properties from which a DCM real-time routing algorithm could be developed. The DCM real-time routing algorithm is described in Chapter 3.

Chapter 3

Dynamic Connection Management(DCM)

In the previous chapters we presented a major problem associated with GPC schemes, i.e. their lack of *flexibility*, and motivated the need for *flexibility* in these schemes. This motivation was, in essence, based on the inability of the current schemes to adapt to the dynamics of the client's "runtime" demands or to the dynamics of the network state. In this chapter we present a GPC scheme that addresses this problem by supporting the modification of traffic parameters, performance parameters, and routes of an existing connection under *global* or *local* control subject to a modification contract that specifies the extent of disruption to be experienced by the client during this modification. *Local* control is the ability of the network to modify the traffic parameters, performance parameters, and route of a *portion*[1] of an existing connection, whereas *global* control refers to the modification of the entire connection. This scheme is one component of a complete solution that we refer to as *Dynamic Connection Management* (DCM), which is composed of the *DCM scheme* and the *DCM policies.*

The DCM scheme is based on the Tenet real-time scheme and extends this scheme by providing *flexibility* to the Tenet scheme. The DCM scheme is the collection of algorithms and mechanisms that permit the network to dynamically modify connection parameters and routes. The modification of a connection is a procedural abstraction whereby a guaranteed performance connection with the new performance and traffic parameters (referred to as the *alternate* connection) is established, the client's traffic is moved from the current real-time connection (referred to as the *primary* connection) to the alternate connection, and then the primary connection is removed. The movement of traffic from the primary to the alternate connection is referred to as the *transition* from the primary to the alternate connection. Modifications can also be applied to change the route of a connection. Furthermore, the

[1]This portion of the connection may contain as many as $N-1$ links in an N link connection; or as few as 1 link.

DCM scheme allows the fast establishment or modification of a channel, permits a finer granularity of control to be exercised on a channel, and supports transparency during the transition from the primary to the alternate channel.

The DCM policies are *rules* that determine if a real-time connection is to be modified, and the new values of its parameters. These rules may examine the network's or client's state data to determine if a modification should take place. If this modification should be effected, then the policies supply the modified parameters and routes. The DCM policies are usually implemented as management applications, and will not be addressed in this thesis.

In this chapter, the DCM scheme can be described from three viewpoints: the DCM modification contract, the DCM algorithms, and the DCM mechanisms. As the DCM scheme is an extension of the Tenet scheme, the Tenet scheme will be first described from the same viewpoints.

3.1 The Tenet Scheme

In this section, we give a brief overview of the current version of the Tenet resource management algorithms [21]. We describe three aspects of the scheme: the Tenet performance contracts, which define the service abstraction; the Tenet mechanisms, which constitute a distributed connection establishment procedure; and the Tenet algorithms, which consist of the service discipline at the switches and the admission control tests.

3.1.1 The Tenet Performance Contract

The Tenet algorithms are based on a communication abstraction called a real-time *channel* [22, 52]. A real-time channel is a network connection associated with pre-specified traffic and performance parameters. The parameters are provided by the clients, who specify their traffic characteristics and performance requirements. The performance guarantees in the Tenet scheme refer to bounds on *throughput*, *delay*, *delay jitter*, and *loss rate due to buffer overflows.* The traffic specification consists of four parameters:

- $Xmin$ - the minimum packet inter-arrival time;
- $Xave$ - the minimum average packet inter-arrival time over an an averaging interval;
- I - the averaging interval;
- $Smax$ - the maximum packet size.

The first three parameters belong to the set of positive real numbers, while the fourth parameter belongs to the set of positive integers. The four performance parameters by which clients describe their requirements are:

- $\overline{D}$ - the maximum delay permissible from the source to the destination;
- Z - the minimum probability that the delay of the packet is smaller than the delay bound, $\overline{D}$;
- W - the minimum probability of no buffer overflow;
- $\overline{J}$ - the maximum delay jitter[2].

The delay violation probability specification allows the client to indicate if a *deterministic* or *statistical* channel is desired. If the value of Z is one, this indicates that the delay of the packet will always be less than the delay bound $\overline{D}$. This is considered a *hard* or *deterministic* guarantee. If the value of Z is less than one, the channel is considered a *statistical* channel. The *throughput* guarantee is obtained from the traffic characteristics of the client as the network, after accepting the channel, agrees to absorb the load produced by the client. The *delay*, *delay jitter*, and *loss rate due to buffer overflows* guarantees are obtained from the performance specifications as the network agrees to deliver packets within the specified performance parameters.

The service abstraction defines a contractual relationship between the network and the client: once the channel is established, the network guarantees that, in the absence of network failures, it will meet the specified performance requirements of the client, provided that the client obeys its worst-case traffic specification.

3.1.2 The Tenet Mechanisms

A channel needs to be *established* before data can be transferred. This channel establishment is achieved in the following manner: a real-time client specifies its traffic characteristics and end-to-end performance requirements to the network; the network determines the most suitable route for a channel with these traffic characteristics and performance requirements; it then translates the end-to-end parameters into local parameters at each node, and attempts to reserve resources by conducting resource reservation tests at these nodes accordingly. This is done in a distributed manner during a round-trip communication.

On the forward pass of the channel establishment round trip, call admission tests are conducted and resources are reserved to get the best possible level of local performance so as to ensure that resource deficiencies further along the path can be accommodated. This process continues along each node until

[2]In this case jitter is defined as the difference between the delays experienced by any two packets on the same connection.

the destination node is reached or an intermediate node rejects the channel. At the end of the forward pass, the destination summarizes the information collected along the path and determines if all of the end-to-end performance bounds obtained by reserving resources during the forward trip are better than the corresponding client requirements. If so, on the reverse pass, the resources reserved during the forward pass are reduced or relaxed so that only the necessary amounts of resources are committed.

These call admission or resource reservation algorithms are used during this channel establishment to determine if the needed resources can be reserved and, if available, to reserve the resources, thereby ensuring that the guarantees made to the clients can be met. Resources are only reserved inasmuch as they do not cause the violation of the guarantees made to the other clients. These algorithms are based on the service disciplines at each switch; therefore, after the establishment phase, *a priori* end-to-end performance guarantees can be offered to the client.

3.1.3 The Tenet Algorithms

In order to provide performance guarantees, two levels of controls are needed: at the connection level, channel admission control algorithms reserve resources for each of the connections and limit the maximum utilization of the network by real-time traffic; at the packet level, the service discipline at each of the switches determines the multiplexing policy and allocates resources to different connections according to their reservations.

As shown in [14, 20, 57, 55], many service disciplines can be used to provide real-time service. However, different service disciplines require different admission control algorithms. For the purpose of this paper, we assume that the service discipline used at the switches is Rate-Controlled Static Priority [56, 55] or RCSP. In this section, we first briefly summarize the properties of the RCSP service discipline, then give the corresponding admission control tests.

RCSP is a service discipline proposed to achieve both *flexibility* in terms of allocating service priorities and bandwidth resources to different connections, and *simplicity* in terms of high speed implementation. As shown in Fig. 3.1, an RCSP server has two components: a rate controller and a static-priority scheduler.

The rate controller shapes the input traffic from each channel into the specified traffic pattern by assigning an eligibility time to each packet; the static-priority scheduler orders the transmission of the eligible packets from all channels. A rate controller is a set of traffic regulators, each associated with a channel traversing the switch. The regulators in RCSP can be either *rate-jitter* or *delay-jitter* controlling regulators. A rate-jitter controlling regulator controls rate jitter[3] by partially reconstructing the traffic pattern, while a delay-jitter controlling regulator controls the delay jitter by fully reconstructing the

[3]Rate jitter is defined as the maximum number of packets that can be present in a jitter averaging interval.

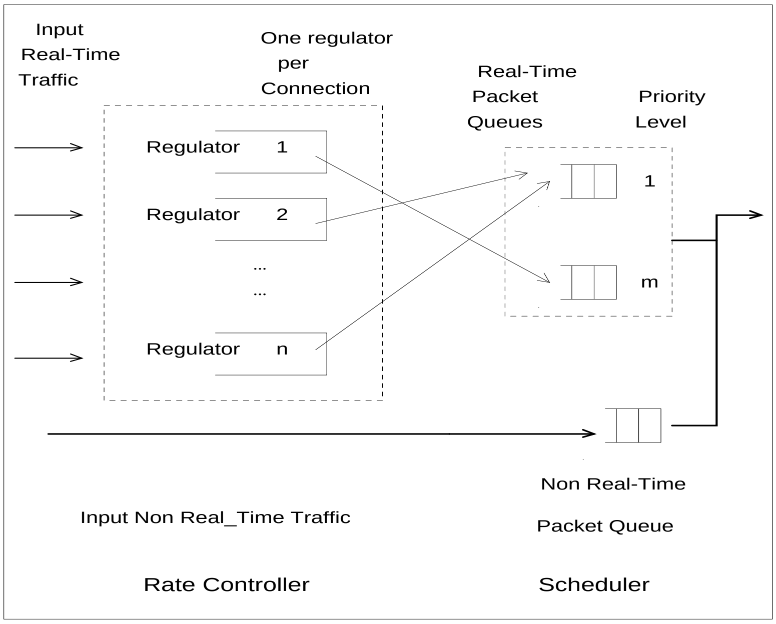

Figure 3.1: Rate-Controlled Static-Priority Queuing

traffic pattern. The regulator achieves this by assigning to each packet an eligibility time and holding the packet until it becomes eligible before passing it to the static-priority scheduler. In controlling the channels admitted into the switch and shaping traffic patterns to conform to the specified traffic requirements, the channels can obtain their previously specified throughput requirements.

The static-priority scheduler consists of a fixed number of prioritized real-time packet queues and a single non-real-time queue. Associated with each priority queue is a delay bound; the packets in the topmost priority queue (denoted as priority level one) have the lowest delay bound. A channel is assigned a priority level during the channel establishment phase, and that level is usually maintained for the duration of the session. Multiple connections can be assigned to the same priority level. By restricting the number of channels at each priority level using the admission tests, the queuing time of each packet at a priority level is guaranteed to be less than or equal to the delay bound associated with that level. The scheduler services the packets as follows: the next packet transmitted is always the packet at the head of the highest-priority non-empty queue, and non-real-time packets are transmitted only when there are no real-time packets in the priority queues. The transmission of a lower-priority packet is not preempted by the arrival of a higher-priority packet.

As stated previously, a single RCSP server can guarantee a number of local delay bounds to different connections. When the local node receives an establishment request, it determines if enough bandwidth and schedulability resources can be reserved to ensure the satisfaction of throughput and delay bound guarantees. Resources are reserved according to the results of Test 1 given below.

Test 1: Let $\overline{d_i^1}, \overline{d_i^2}, \ldots, \overline{d_i^n}$ $(\overline{d_i^1} < \overline{d_i^2} < \cdots < \overline{d_i^n})$ *be the delay bounds associated with each of the n priority levels, respectively, at switch i. Let* C_q *be the the set of connections that are established and assigned level q* $(1 \leq q \leq n)$*, and the* j^{th} *connection within* C_q *has the traffic specification* $(Xmin_j^q, Xave_j^q, I_j^q, Smax_j^q)$*. Assume that the link speed is l, and the size of the largest packet that can be transmitted onto the link is* $\overline{Smax}$ *. A new connection with the traffic specification* $(Xmin_{new}, Xave_{new}, I_{new}, Smax_{new})$ *can be assigned to level m, or be assigned a local delay bound* $\overline{d_i^m}$*, if the following inequality holds:*

$$\sum_{q=1}^{m'} \sum_{j \in C_q} \lceil \frac{\overline{d_i^{m'}}}{Xmin_j^q} \rceil Smax_j^q + \lceil \frac{\overline{d_i^{m'}}}{Xmin_{new}} \rceil Smax_{new} + \overline{Smax} \leq \overline{d_i^{m'}} l \qquad m' = m, \cdots, n \tag{3.1}$$

Intuitively, the longest waiting time in the scheduler for a level-m' packet corresponds to the case in which a lower-priority packet is being transmitted when the packet arrives at the scheduler, and is followed immediately by the longest possible transmission of packets with higher or equal priorities. Test 1 bounds this longest waiting time to be less than $\overline{d_i^{m'}}$ for $m' = m, \cdots, n$, which are the priority levels that can be affected by placing a new connection in priority level m (level 1 has the highest priority).

RCSP servers also hold packets to ensure traffic smoothness and bounded delay-jitter properties in a network of switches. The following gives the delay property for a connection traversing a tandem of RCSP switches [4].

Delay Property: Let $\overline{d}_{1,j}, \cdots, \overline{d}_{i,j}$ *be the local delay bounds for the first i switches along the path traversed by connection j,* $\pi_{i-1,i}$ *be the propagation delay from switch* $i-1$ *to switch i, and* $D_{i,j,k}$ *be the delay experienced by the* k^{th} *packet on connection j from switch* 1 *to switch i; the following properties hold for any k:*

$$\sum_{i'=1}^{i-1} (\overline{d}_{i',j} + \pi_{i',i'+1}) \leq D_{i,j,k} \leq \sum_{i'=1}^{i-1} (\overline{d}_{i',j} + \pi_{i',i'+1}) + \overline{d}_{i,j} \tag{3.2}$$

The property gives the upper and lower bounds on the delay for any packets traversing a path of RCSP switches. The end-to-end delay jitter is bounded by the local delay bound of the last switch along the path.

To ensure enough buffers are reserved so that the performance guarantees are not violated, the following local condition must be met at switch i.

Test2:

[4]Of the two types of RCSP servers only a delay-jitter controlling RCSP server has this delay property [55]. In this dissertation only delay-jitter controlling RCSP servers are used.

$$R_{bu} + \lceil\frac{\overline{d}_{i-1,j}}{Xmin}\rceil \times Smax + \lceil\frac{\overline{d}_{i,j}}{Xmin}\rceil \times Smax \leq B \qquad i = 1, \cdots, n \tag{3.3}$$

where $\overline{d}_{i-1,j}$ *and* $\overline{d}_{i,j}$ *are the local delay bounds for the connection at the* $(i-1)_{th}$ *and* i_{th} *switches along the path, respectively,* R_{bu} *is the current buffer space occupied (in bits) and* B *is the maximum buffer space (in bits) allotted.*

Note that the buffer space depends on the delay bound in the previous switch. This is due to fact that an RCSP switch also holds packets, and the longest time a packet from connection j is held in switch i is $d_{i-1,j}$ where $\overline{d}_{0,j} = 0$.

3.2 The Dynamic Connection Management Scheme

3.2.1 The DCM Modification Contract

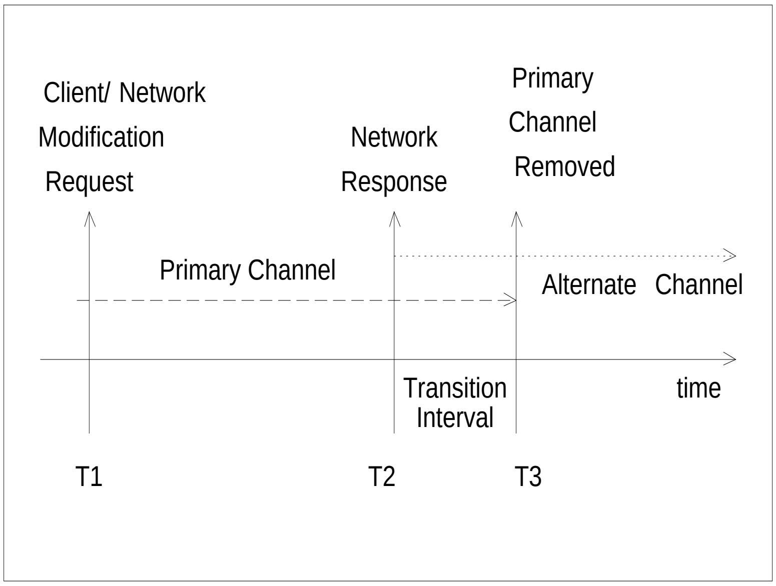

Figure 3.2: Request/Response Paradigm

A request for modification is governed by the request/response paradigm depicted in Figure 3.2. In this paradigm, requests can be either client-initiated or network-initiated. In a client-initiated request, an application or higher level protocol entity makes a modification request to the network services manager, whereas in a network-initiated request the DCM policy manager, using the current network state and client demands, can decide that a modification is needed and submit a request to the network service manager. This is shown in Fig. 3.2 at time T1. In this thesis we will normally use the term "client" to refer to both an application or a policy manager as these are the clients that use the network services in the DCM scheme. In most cases the context will indicate to the reader to which, if not both, of the client types we are referring. Upon receiving a request, the network service manager uses the DCM scheme to modify the relevant channels. It should be noted that the DCM scheme can also be used to establish a connection as well as to modify it. After the request is made the network returns a response, *accepted* or *denied* (at time T2 in Fig. 3.2) based on the contents of the request and the current real-time network load. If the request has been accepted, the client can then begin sending packets using the new traffic characteristics, if applicable[5], and expecting that the performances guaranteed on both the primary and alternate channels are met[6]. It should be noted that the primary and alternate channels exist simultaneously for a short interval of time, *the transition interval* (between T2 and T3 in Fig. 3.2), and then the primary is removed. After this interval only the alternate channel will exist. Currently the DCM scheme only supports the modification of deterministic services.

In DCM there are contractual obligations made to the client that determine the extent of the disruption that will be experienced by the client due to the transition from the primary to the alternate channel. There are two types of DCM modification contracts:

1. The *No-Violation contract*, which states that *no performance guarantees will be violated* during the transition from the primary to the alternate channel.

2. The *Bounded-Violation contract*, which states that *a bounded number*[7] *of performance violations can occur* during the transition from the primary to the alternate channel.

There are three types of performance violations that may occur:

- a *delay bound* violation occurs when a client, sending traffic according to its traffic characteristics, has at least one packet that exceeds the delay bound $\overline{D}$ at the destination node;

- a *delay jitter bound* violation occurs when a client, sending traffic according to its traffic characteristics, has at least one packet that exceeds its delay jitter bound $\overline{J}$ at the destination node; and

- a *packet ordering* violation occurs when a client, sending traffic according to its traffic characteristics, receives packets out of sequence.

These violations may occur singly, or multiple violations can occur simultaneously. The first type of DCM modification contract ensures that none of the three performance violations will occur during the

[5]There may be network-initiated requests where only the route is changed, hence there are no new traffic characteristics.

[6]This is to say that packets traversing the primary channel will meet the performance guarantees corresponding to the primary channel, and packets traversing the alternate channel will have their alternate channel performance guarantees met, subject to the modification contract explained below.

[7]The number of performance violations is specified as a packet count; however, it will be converted into a unit more easily understood by the human client (i.e., frames, messages, etc.).

transition to an alternate channel. This contract may be explicitly requested by the client before channel parameter modification or implicitly demanded by the policy manager before channel route modification. Route modification is usually done by the network and must be totally transparent to the client; hence there must be no performance violations. Reroutings may be done directly for network administrative or management purposes, or indirectly due to a client's performance parameter modification request.

There is an intrinsic condition that must be satisfied to avoid performance violations. To see this, let us consider the following case. Assume the k^{th} packet to be the last packet transmitted on the primary channel. Let $\overline{D^p}$, $\overline{D^a}$ be the end-to-end delay bounds of the rerouted connection on the primary and alternate paths, respectively. Also, let s_k, r_k, D_k and s_{k+1}, r_{k+1}, D_{k+1} be the sending time, receiving time, end-to-end delay, for the k^{th} and $(k+1)^{th}$ packets, respectively. We have $s_k + D_k = r_k$, and $s_{k+1} + D_{k+1} = r_{k+1}$. To ensure in-order delivery, we need to have $r_k < r_{k+1}$, i.e. $s_k + D_k < s_{k+1} + D_{k+1}$. Rearranging the terms we have

$$D_k - (s_{k+1} - s_k) < D_{k+1} \tag{3.4}$$

We assume that the k^{th} packet traverses the primary route and the $(k+1)^{th}$ packet traverses the alternate route. We also know that $D_k \leq \overline{D^p}_{path}$, where $\overline{D^p}_{path}$ is the upper bound on the delay of packets along the primary path and that the traffic has to satisfy the traffic constraint, or $s_{k+1} - s_k \geq Xmin^a$. Since (3.4) has to hold for any values of D_k and D_{k+1} that satisfy delay and jitter bounds, we choose the most rigorous situation[8] to get the following:

$$\overline{D^p}_{path} - Xmin^a < \overline{D^a}_{path} - \overline{J^a}_{path} \tag{3.5}$$

where $\overline{D^a}_{path}$ is the the upper bound on the delay of packets along the alternate path, and $\overline{J^a}_{path}$ is the maximum jitter experienced by packets along the alternate path. Hence $\overline{D^a}_{path} - \overline{J^a}_{path}$ is the minimum amount of time that packets traversing the alternate path can experience between the source and the destination (i.e., it is the lower bound on the delay of packets along the alternate path; this is due to the delay-jitter rate control RCSP servers); thus, $\overline{D^a}_{path} - \overline{J^a}_{path} \leq D_{k+1}$. Since the modification contract must be made before establishing the alternate channel, the values of $\overline{D^a}_{path}$ and $\overline{J^a}_{path}$ will not be known. However, we do know that the following

$$\overline{D^a}_{path} \leq \overline{D^a} \tag{3.6}$$

must be true to ensure the the delay bound guarantees are met on the alternate channel. Thus the necessary condition[9] for a No-Violation contract is

$$\overline{D^p}_{path} - Xmin^a < \overline{D^a} \tag{3.7}$$

[8] Where the largest values are chosen for the variables on the left hand side of equation 3.4 and the smallest for those on the right hand side of the equation.

[9] We assume here that the traffic characteristics of the alternate channel apply at the instant that the client is notified of the acceptance.

While the No-Violation contract constrains the ranges of the parameter modifications, some clients can tolerate the violation of these guarantees during a transition provided that the effect is bounded. The performance violations that can occur are those of delay, delay jitter, and packet ordering, and the violation bound is an upper bound on the number of packets exceeding their required delay or delay jitter bounds, or the number of packets that are received out of order at the destination during the transition interval. This contract is useful, as we believe most clients will expect a slight disruption in service upon modification, and will be ready to accept it as long as it is bounded.

In the case of a Bounded-Violation contract, we remove the parameter constraints and specify an upper bound on the number of packets that will exceed the delay or delay jitter bounds of the alternate channel, or arrive out-of-sequence at the destination during the transition interval. If condition (3.7) is not satisfied, the number of packets that can arrive out-of-sequence or exceed delay bounds is bounded by

$$\lceil \frac{\overline{D^p}_{path} - Xmin^a - \overline{D^a}_{path} + \overline{J^a}_{path}}{Xmin^a} \rceil + 1. \tag{3.8}$$

As a performance violation bounds must be supplied to the requester before the channel is modified, the value of this bound will be made known to the client before the channel modification is attempted, and the client can then decide if the modification is worth attempting. Equation 3.8 determines the number of packets that will have performance violations but also requires information that can only be obtained after modification of the channel (i.e., $\overline{D^a}_{path}$ and $\overline{J^a}_{path}$). Therefore, before modification, the number of packets with performance bounds is determined by the equation:

$$\lceil \frac{\overline{D^p}_{path} - Xmin^a - D^{min}{}_{path}}{Xmin^a} \rceil, \tag{3.9}$$

where $D^{min}{}_{path}$ is the sum of the propagation and transmission delay on the shortest path between the source and the destination. This equation is a much looser version of equation 3.8 as it uses the $D^{min}{}_{path}$ term, which is the absolute minimum delay between the source and the destination, and will provide a high upper bound on the number of packets with performance violations. If the client wishes to continue with the modification, then after establishment the client can be informed of the tighter bound stated in equation 3.8. This violation bound is the same for all performance violations, as it is based on the maximum number of primary channel packets that can be in transit during the transition from the primary to the alternate channel.

3.2.2 The DCM Algorithms

In this sub-section we will present the high level functionalities of the three DCM algorithms: the channel administration algorithm, the routing algorithm, and the transition algorithm, followed by a detailed discussion of each of them. The key function of these algorithms is to provide the support needed to modify parameters and routes under the constraints of *resource sharing* and of *the modification contracts*. These two constraints motivate three specific functions, each of which is provided in an algorithm.

Alternate channel establishment can be examined under two scenarios, i.e., *no resource sharing* and *resource sharing*. Under the *no resource sharing* scenario, an alternate channel is established along a route that is completely disjoint from that of the primary channel or the alternate channel traverses links that are common with those of the primary channel but does not share any of the resources previously reserved by the primary channel. Under the *resource sharing* scenario, an alternate channel is established along a route which traverses some links that are common with those of the primary channel and shares resources along all of the common links[10]. We envision that resource sharing may be desirable[11] as we expect a significant number of instances in which a very large channel (i.e., very resource demanding) or multiple smaller channels are being rerouted or enhanced, and the resources required to accommodate these requests, especially during the transition interval, can only be made available using resource sharing. The channel administration algorithm provides the admission control tests and some additional constraints needed to support both of these scenarios. It should be mentioned that the decision to utilize resource sharing is entirely policy-dependent, and the algorithm merely provides the capability without imposing any judgment as to when it is used.

In determining a suitable route for the alternate channel, the routing algorithm must be able to reflect the inclusion or exclusion of the resource sharing factor. If resource sharing is not accommodated, the resources reserved by the primary channel are not considered by the routing algorithm in determining an alternate route. If resource sharing is accommodated, the routing algorithm compensates for the resources reserved by the primary channel, i.e., by "virtually" removing the resources reserved by the primary channel from its routing database before calculating the alternate route. Thus, the alternate route chosen may have to share previously reserved resources if there are common links. The decision to accommodate resource sharing is a policy decision; only the routing mechanism, which must support either policy, will be addressed in this work.

The modification contracts discussed in Section 3.2.1 present the performance parameter constraints needed to support the "No-Violation" and "Bounded-Violation" contracts; however, in the "No-Violation" contract additional support is needed to prevent packet-ordering or delay bound performance violations. The transition algorithm described below provides the additional buffers and the packet reordering mechanism needed to support this contract.

Table 3.1 summarizes the discussion above. The constraints of resource sharing and the modification contracts are expanded, and their effects on the three DCM algorithms are presented. There are two

[10]It may be that the alternate and primary route are the same.

[11]It would increase the utilization of the network.

Constraints	Channel Admission Algorithm	Routing Algorithm	Transition Algorithm
NV RS	Test Set (1,2,3,4)	Adjust resources before route calculation	Used
NV NRS	Test Set (1,2)	Do not adjust resources before route calculation	Used
BV RS	Test Set (1,2,3,4)	Adjust resources before route calculation	Not Used
BV NRS	Test Set (1,2)	Do not adjust resources before route calculation	Not Used

Table 3.1: Impact of Constraints on DCM Algorithms

types of modification contracts, a *No-Violation contract (NV)*, and a *Bounded-Violation contract (BV)* . The resource sharing constraint is reflected in two states, *Resource Sharing (RS)* and *No Resource Sharing (NRS)* . There are four tests associated with the Channel Administration algorithm; the first two tests, Test 1 and 2 presented in Section 3.1.3, are used for admission control on non-shared resources, while Test 3 and 4 (to be presented in Section 3.2.2), are used for admission control on shared resources. Along a path on which resource sharing occurs there may be both shared links and unshared links; hence all of the tests (i.e., Tests 1, 2, 3, and 4) may be used in that establishment attempt.

The DCM Channel Administration Algorithm

The DCM scheme has the same procedural format as the Tenet scheme, in that an establishment or modification message proceeds along the nodes traversing the path, and admission tests are conducted to determine if the new channel can be established or modified and to reserve the appropriate resources for this channel. In the DCM scheme, the Tenet channel administration algorithm has been supplemented by the DCM channel administration algorithm.

The goal of the DCM channel administration algorithm is to establish an alternate channel, conforming to the specified traffic and performance parameters, between a source and a destination host. This alternate channel is established in the presence of a primary channel on which the client is currently active. The establishment entails the decision as to the acceptance or rejection of the client's request subject to resource availability; the algorithm must reserve the appropriate resources if they are available, so that an *a priori* guarantee is made. In the establishment of an alternate channel we can choose not to utilize or to utilize resource sharing. Both scenarios are examined below.

As discussed previously, in a *no resource sharing* scenario the alternate channel is completely resource independent from the primary channel and the admissions tests to be applied at each link are those used in the establishment of a primary channel, i.e., Test 1 and Test 2. There is, however, one difference: the transparency procedure (discussed in Section 3.2.3) is used to ensure that the interface the client sees of the channel after the modification is the same as that seen before the modification.

In the *resource sharing* scenario, the alternate channel is resource dependent on the primary route, and shares resources along one or more of the links that comprise the primary route. In this scenario the admissions tests applied to the common links, Test 3 and 4, reserve resources for the larger (in terms of resource reservations) of the two channels. Test 1 and 2 are still applied to the links that are not common to both routes. Test 3 and 4 are modifications of Tests 1 and 2, respectively, and take into consideration resources that are already reserved for the primary channel at that link, to ensure that there is no duplication of resources. If the primary channel has acquired resources that are greater than those of the alternate channel, these tests need not be applied. If Test 3 and 4 are successful, we have guaranteed that enough resources are available for the higher performance channel but not for both of the channels; therefore, we need to avoid the situation in which packets from both channels arrive at the link simultaneously, as resources are not reserved for both channels. This is achieved by using the delay jitter control properties of the RCSP server, that is, by properly setting the local delay bounds parameters along the paths of both the primary and the alternate channels. Notice that, in a delay-jitter controlled network, the delay of a packet from a source to a switch does not have only an upper bound, but also a lower bound, and the difference between the two bounds, which is the maximum delay jitter, can be tuned by properly setting the local delay bounds [56, 57]. Assume that the shared link is an output link of switch i, that the upper bounds and lower bounds of delay from the source to the i th switch for packets of the primary and alternate channels are denoted by $\overline{D_i^{upper,p}}$, $\overline{D_i^{lower,p}}$[12], $\overline{D_i^{upper,a}}$ and $\overline{D_i^{lower,a}}$ respectively. To ensure that the packets arriving at the shared link obey the traffic specification, the following condition must be satisfied:

$$\overline{D_i^{upper,p}} \leq D_i^{lower,a} \tag{3.10}$$

If both of these actions, passing the establishment tests and fulfilling the above delay bounds condition, can be done successfully, the channel can be accepted; otherwise, it is rejected.

Resource Sharing Admissions Control Tests

These modified tests are only applied to the common links if any of the performance requirements of the alternate channel are *greater* than those of the primary channel. When at least one of the conditions (provided below) holds, the performance needs of the alternate channel are *greater* than those of the primary channel. These conditions correspond to the throughput, delay, and delay jitter performance of the channels, respectively, and are:

- $\frac{Smax^a}{Xmin^a} \geq \frac{Smax^p}{Xmin^p}$

[12]In keeping with our convention, the "bar" over $D_i^{lower,p}$ indicates a bound on this lower delay.

- $\overline{D}^a < \overline{D}^p$
- $\overline{J}^a < \overline{J}^p$

To ensure that the transition from the primary to the alternate channels is as smooth as possible, it is necessary to retain ample resources so that packets from either the primary or alternate channels can meet their requirements. This can be achieved by a judicious choice of parameters upon which resource reservation at this common link will be based. In Test 3 below, an initial adjustment is made to virtually remove the resources currently reserved for the primary channel, and then resources are reserved for the composite channel defined by equations (3.11), (3.12), and (3.13)[13]. After this adjustment, the resources must be available to ensure that packets from both channels meet their obligations. This is accomplished by choosing the appropriate $Xmin$ and $Smax$ parameters from among those of the primary and alternate channels, and ensuring that on that common link the local delay bound of the alternate channel is always less than or equal to that of the primary channel. In the event that any performance index for the alternate channel is greater than the same index for the primary channel, the composite values of $Xmin$ and $Smax$ to be used in the admission test, and the local delay conditions are given by:

$$Xmin^c = min(Xmin^a, Xmin^p) \tag{3.11}$$

$$Smax^c = max(Smax^a, Smax^p) \tag{3.12}$$

$$\overline{d^c_{\hat{i}}} = \overline{d^a_{\hat{i}}} \leq \overline{d^p_{\hat{i}}} \tag{3.13}$$

If all of the performance indices of the alternate channel are less than those of the primary channel, no admission test need be applied, as sufficient resources for the alternate channel have already been reserved. As the resources reserved by Test 3 below ensure that the primary channel packets as well as the alternate channel packets meet their obligations, they may be in excess of those needed for the alternate channel. These excess resources are only present during the transitional period, and will be recovered by the network upon the tear down of the primary channel.

Test 3: For an alternate channel request with the traffic specification ($Xmin^a, Xave^a, I^a, Smax^a$) and a local delay bound requirement of $\overline{d^{a,k}}$, and for a primary channel with specification ($Xmin^p, Xave^p, I^p, Smax^p$) and a local delay bound $\overline{d^{p,m}}$, first the resources are adjusted to "virtually" remove the primary channel and then Test 1 is applied.

[13]The subscript $\hat{i}$ indicates that link i is common.

Adjustment:

$$R_{ba,m'} = \sum_{q=1}^{m'} \sum_{j \in C_q} \lceil \frac{\overline{d^{m'}}}{Xmin_j^q} \rceil Smax_j^q + \overline{Smax} - \lceil \frac{\overline{d^{p,m'}}}{Xmin^p} \rceil Smax^p \qquad m' = m, \cdots, n. \tag{3.14}$$

If the condition given below can be met:

$$R_{ba,m'} + \lceil \frac{\overline{d^{c,m'}}}{Xmin^c} \rceil \times Smax^c \leq \overline{d^{c,m'}} l \qquad m' = k, \cdots, n \tag{3.15}$$

then the alternate channel request can be accepted.

The buffer resource test modification is of the same form as that of the bandwidth and scheduling test above, but an adjustment must be made to ensure that there is no duplication of previously reserved resources. Again, the reserved resources are adjusted, and the test condition applied to the adjusted resources.

Test 4: For an alternate channel request with the traffic specification ($Xmin^a, Xave^a, I^a, Smax^a$) and a delay bound requirement $\overline{d^a}$, and for a primary channel with a traffic specification ($Xmin^p, Xave^p, I^p, Smax^p$) and a delay requirement $\overline{d^p}$, the adjustment at the ith switch is:

$$R_{bu_{adj}} = R_{bu} - \lceil \frac{\overline{d_{i-1}^p}}{Xmin^p} \rceil Smax^p - \lceil \frac{\overline{d_i^p}}{Xmin^p} \rceil Smax^p \tag{3.16}$$

The condition that needs to be satisfied is :

$$R_{bu_{adj}} + \lceil \frac{max(\overline{d_{i-1}^p}, \overline{d_{i-1}^a})}{Xmin^c} \rceil Smax^c + \lceil \frac{\overline{d_i^p}}{Xmin^c} \rceil Smax^c \leq B \tag{3.17}$$

where R_{bu} is the current buffer space in use (in bits), B *is the maximum buffer size (in bits) allocated to that output link, and $\overline{d_{i-1}^p}$ is the delay bound in the $i-1$ th switch along the route.* It should be noted that $\overline{d_{i-1}^p}$ actually refers to the delay in the switch on the primary route preceding the ith switch along the alternate route. This preceding switch along the primary route may not be the $i-1$th switch on that route, but for the sake of notational brevity we allow this exception.

Note also that this modified test is only performed if the performance indices of the alternate channel are greater than those of the primary channel. In the case where the performance requirements are less restrictive, no resources are released during the establishment phase; rather, the excess resources are reclaimed during the tear down of the primary channel.

The DCM routing algorithm is designed to find an shortest path route based on the constraints imposed by the traffic characteristics, the performance and administrative requirements, and the source/destination host pair. A shortest path route is one that minimizes the total *cost* as defined below. This routing algorithm taken by itself would provide a significant contribution to the Tenet Scheme 1 as the scheme currently uses Internet routing, which considers neither the real-time network load nor the traffic and performance parameters of the channel. The DCM channel administration algorithm obtains from the DCM routing algorithm a route for the specified source/destination host pair that obeys the specified routing constraints. In requesting this route, the values of various traffic, performance, and administrative parameters are required by the routing algorithm. The traffic and performance parameters pertaining to the alternate and primary channel have been previously described. The administrative parameter is used to indicate resource sharing. This parameter can take three values:

- if the parameter value is 0, the routing algorithm assumes that a primary route is needed, and obviously no resource sharing occurs;
- if the parameter value is 1, the routing algorithm determines an alternate route that does not share resources with the primary route;
- if the parameter value is 2, an alternate route that can share resources with the primary route is determined.

The manner in which these administrative requirements are satisfied is explained below.

The DCM Routing Algorithm provides source routing and is based on a modified, constrained, version of the Bellman-Ford algorithm. In a network with N nodes, the fundamental Bellman-Ford algorithm [10] searches for the shortest paths between a specified source and destination node starting from all possible one-hop paths and continuing until all N-2-hop paths have been examined. The goals of our routing algorithm were to maximize throughput, to obtain routes in a timely manner, and to maximize the probability that the route provided will be successfully established (i.e., the route will be established with the traffic and performance specifications given by the client). The routing algorithm calculates a minimal-cost route subject to a delay constraint. The cost of the route is the number of links comprising the route, while the delay constraint ensures that the sum of the delay values of these links is less that the delay bound, $\overline{D}$, required by the client. The delay value attributed to a link is the sum of the minimum queuing delay offered by the node to a real-time channel with these traffic characteristics and the propagation delay along the output link. While the propagation delay is fixed, the queuing delay experienced in the RCSP scheduler is variable, and is dependent on the current channel resource reservations on the corresponding output link and the traffic characteristics of the new channel. This queuing delay is calculated by using the admission tests provided in Section 3.1.3 and Section 3.2.2 to determine the minimum queuing delay that this link can offer a connection with these traffic characteristics. Currently we assume that buffer space is an abundant, although finite, resource and so it is not considered a constraint in the routing algorithm.

The algorithm proceeds with the following steps:

- A directed graph is created in which the nodes correspond to switches and hosts in the network and the edges to the links connecting these switches and hosts. The *weights* attributed to the edges represent the link delay values. These delay values are computed just prior to applying the algorithm, thereby using the most recent link information obtained from routing update messages.

- For the delay bound case the DCM routing algorithm proceeds as follows:

  1. Consecutive searches are performed on all 1, 2, .., N-2-hop paths from the source to the destination node, where N is the number of nodes in the network, until the delay condition $\sum_{l(s,r)} w_l \leq \overline{D}$ is satisfied, where $\overline{D}$ is the delay bound of the channel, w_l is the delay value or *weight* of link l, and *(s,r)* are the links connecting the source s to the destination r. A *constraint* is placed on the number of possible searches by stopping at the *hop level* at which the delay bound condition is first satisfied. This *hop level* is the number of hops from the source to the destination node and is also the cost of the route.

  2. At this *hop level* or cost, the path with the minimum delay value (i.e., $\min\sum_{l(s,r)} w_l$) that meets the delay condition is chosen.

- For the delay jitter bound ($\overline{J}$) case, the following steps are performed:

  1. Assuming a path with n hops, the minimum queuing delay offered by the link incident on the destination node[14], d_n, is first examined. If $d_n \leq \overline{J}$, then the algorithm proceeds, else the channel cannot be accepted, as the delay jitter bound condition cannot be satisfied.

  2. Consecutive searches are performed on all 1, 2, .., N-2-hop paths from the source to the destination node until $\sum_{l=1}^{n-1} w_l \leq \overline{D} - d_n - \pi_n$, where $\overline{D}$ is the delay bound of the channel, and π_n is the propagation delay associated with the last link n.

  3. At this hop level, the path with the minimum delay value that meets the delay condition is chosen.

As we consider bandwidth to be our premium resource, the algorithm seeks to reduce the consumption of this resource by selecting a path with the minimum number of hops so as to maximize the available

[14]We assume that there is only one link incident on the destination node. In the event that there are multiple incident links, the link with the lowest queuing delay value is chosen.

network throughput. This is achieved by limiting the search space or *hop level*, which restricts the number of links or *hops* in the path. Another consequence of limiting this search space is that the computation time of the algorithm is reduced. The probability of successful channel establishment is increased as the algorithm determines the queuing delays based on the traffic characteristics of the channel and the most recent real-time load information.

The administrative constraints are achieved by modifying the weights (i.e., the delay values) associated with the edges (i.e., links) of the graph before applying the algorithm. With an administrative parameter value of 0 and 1, no adjustment is made to the edges of the graphs. Thus, in the presence of a primary channel, i.e., when the parameter is 1, the resources used by the primary channel are not considered. With a value of 2, the primary channel's resources are virtually removed from the edges corresponding to the links comprising the primary route before calculating the weight of the edge.

Routing updates are currently done on a per-channel-establishment basis. Updates are accomplished by having every node broadcast *the load values* of its links to all other nodes. These load values are the amount of bandwidth and delay resources reserved at each priority level in the RCSP server at this node. This broadcast is done upon the establishment of a new channel, after the node has sent the reverse channel establishment message to the previous node on the new channel's route, and following every channel tear down. Upon receiving an update packet, the receiving node updates its local link-state table. If there are no new channel establishments within a specified time interval, link updates are sent by each node to assure other nodes that the link is still active. All route update broadcasts are done along a minimum spanning tree.

The DCM Transition Algorithm

The DCM transition algorithm ensures that the transition from the primary to the alternate channel does not violate the DCM modification contract. It is invoked for a channel with a *No-Violation* modification contract when there is a possibility that packets on the alternate channel may arrive at the destination before the last packet on the primary channel. This packet mis-ordering situation may arise even when condition 3.7 is met. In this case, transition buffers need to be reserved at the destination and the re-sequencing of packets needs to be performed. During channel establishment some buffers have been reserved to ensure that all performance guarantees are met; however, additional buffers may need to be reserved to accommodate out-of-sequence packets.

Packets along the alternate route may arrive at the destination before packets along the primary route only when the maximum delay of packets traversing the primary channel is greater than the minimum delay of packets traversing the alternate channel. This condition is:

$$\overline{D^p}_{path} - Xmin^a + \overline{J^a}_{path} \geq \overline{D^a}_{path} \tag{3.18}$$

If this condition 3.18 is satisfied, then the additional buffers needed are:

$$\left(\lceil \frac{\overline{D^p}_{path} - Xmin^a - \overline{D^a}_{path} + \overline{J^a}_{path}}{Xmin^a} \rceil + 1 \right) Smax^a \tag{3.19}$$

During the transition, packets arriving earlier on the alternate channel will be held in these buffers until all packets from the primary channel have arrived and have been passed to the receiver.

In the case of a connection requesting a delay-jitter bound, the channel administration algorithm makes the delay bound at the last switch equal to the delay jitter bound. If $\overline{J}^a \neq \overline{J}^p$, contract violations can be avoided by maintaining a delay jitter equal to $\min(\overline{J}^p, \overline{J}^a)$. If condition (3.18) holds, the reserved buffers (including the transition buffers) will be used to ensure that delay-jitter performance guarantees are not violated during the transition. All out-of-sequence packets on the alternate channel will be buffered at the destination and passed up to the client at the appropriate time. In the previously discussed delay bound case, upon arrival of all packets on the primary channel the buffered out-of-sequence packets are all passed up to the client immediately. However, to preserve the exact traffic pattern in the delay jitter bound case, the out-of-sequence packets will be passed up to the client at the appropriate times. The appropriate time, t_k, for packet k is

$$t_k = src_k + \overline{D^a} - \overline{J^a}, \tag{3.20}$$

where src_k is the source time stamp on the packet, and $\overline{D^a}$ and $\overline{J^a}$ are the delay bounds and delay jitter bounds on the alternate channel. As mentioned previously, if the No-violation contract is desired, the tight coupling of delay bounds and delay-jitter bounds in our scheme necessitates that the conditions specified by equation (3.7) be met to ensure that there are no delay jitter performance violations.

3.2.3 The DCM Mechanisms

As mentioned in the previous section, the DCM mechanisms employ the same procedural format as the Tenet scheme for the establishment of a channel. The modification of a channel also follows a similar format as that of the establishment, in that two passes are required for establishment or modification, and resources are reserved on the forward pass and relaxed on the reverse pass. However, the procedure has been enhanced to reduce the response time of the DCM scheme (i.e., the overall delay experienced by the client between the time of its request and that of the network service manager's response).

In this subsection we will address the mechanisms unique to the DCM scheme that allow the fast establishment or modification of a channel, that permit the granularity of control, and that support transparency during transition from the primary to the alternate channel.

The response time of the DCM scheme is the time interval between an establishment or modification request and the response of the network. This interval is dependent on the client's request, the current network load, and the establishment procedure, and is commonly referred to as the establishment time[15]. With channel establishment we have the following two cases: if the client requests a modification that does not increase the level of any of the performance parameters or require a change in the route of the channel, then the response is an immediate **acceptance**, and, if applicable, the traffic characteristics of the channel can be immediately modified to reflect this response; if the modification request reflects an increase in the level of at least one of the performance parameters or a change in the route of the channel, then the response can be an **acceptance** or a **rejection** of the request. If the response is an acceptance, then at the instant the response is passed to the client the required resources are available, and the client can now begin to use these resources.

The response time should be as short as possible for client satisfaction. This time is dependent on

- the accuracy of the routing algorithm (i.e., the ability of the algorithm to provide a route that will permit the establishment of the channel requested by this client), and,
- the round trip delay (i.e., the sum of the transmission, propagation, and queuing delays) of the establishment message along the route from the source to the destination.

In order to reduce the establishment time, these two dependencies must be examined.

In DCM we attempt to reduce the portion of the establishment time dependent on the accuracy of the routing algorithm by (1) having the routing algorithm utilize the most recent real-time network state information as well as the traffic and performance requirements of the client[16] to determine an available route before establishing the channel, and (2) exploiting the *time value* of the network state information so as to bypass unavailable links (discussed later in this section) during channel establishment. The DCM mechanisms employ a modified version of the establishment procedure used by the Tenet scheme that exploits the *time value* of the network state information to achieve a greater success probability.

In order to guarantee that the route selected by the routing algorithm will provide the greatest probability of success, the routing algorithm uses the most recent network state information in its database and the traffic and performance requirements of the channel in computing a route. The accuracy of this network state information is based entirely on the routing update mechanism. In our model, route update information is disseminated upon the establishment and tear down of a channel

[15]In this thesis, the interval is referred to as the *primary establishment time*, in the case of the response time to an initial request, and as the *alternate establishment time* in the case of the response time to an alternate channel request.

[16]The current Tenet scheme utilizes Internet routing which does not directly utilize any network or client real-time information.

and periodically if there are infrequent channel establishments or modifications. Currently, these route updates are sent via a minimum spanning tree connecting all nodes in the network [17].

The value of network state information decreases in time if it is not updated. At the time of its shipment, an update packet contains the exact state of the node[18] from which it was generated; as it is sent to nodes further away from this source node, the network state changes and the information provided in the packet is a less accurate description of the state of the source node. Beyond a certain amount of time the information contained in a routing update packet may be entirely inaccurate. As a channel establishment message moves from the source node towards the destination node, it encounters nodes that have received more recent network state information from the destination node. Thus, the time value of the information pertaining to the route along which the establishment message is traveling is increasing. This indicates that the nodes have more precise information on the state of the network along that route as the establishment message moves towards the destination. If the establishment message encounters a link which has insufficient resources to accommodate the request, it returns to the node preceding this unavailable link and requests a new route from this node to the destination. This node, which has more recent information than the source of the establishment package, computes the route from itself to the destination. This computation takes into consideration the resources that were reserved before the unavailable link, and the traffic and performance parameters contained in the establishment message. Each node that attempts to route the establishment packet because of an unavailable link knows the previously reserved portion of the route and removes these links (with the reserved resources) from the graph, before applying the routing algorithm, thereby preventing looping in the path. This addition to the establishment procedure (*a.k.a. intelligent restart*) should reduce the establishment time of a channel, and can be useful also in the case of link failures where the establishment message can be immediately rerouted to avoid the failed link. An example of the new establishment procedure is shown in Figure 3.3, where a channel establishment is being attempted between the source node **S** and the destination node **D**. The initial route chosen for this attempt is route 1 (labeled (**1**)). The admission control test for resource reservation is successful at the first two links in the path but fails at the third link (i.e., the link between intermediate nodes I2 and I3). At this point the establishment procedure returns to the previous intermediate node, I2, and attempts to route the channel to the destination node from that intermediate node. In this case there is a possible route, route 2, hence the establishment message proceeds along this new "sub-route" and the channel is established. If the admission test failed on the second link (between I2 and I3) then route 3 may have been the new sub-route.

[17] While a sufficiently large network will suffer from inconsistent views of its state due to the large delays experienced in the broadcast, for small networks this mechanism is adequate.

[18] The nodes contain information on the state of the links they support.

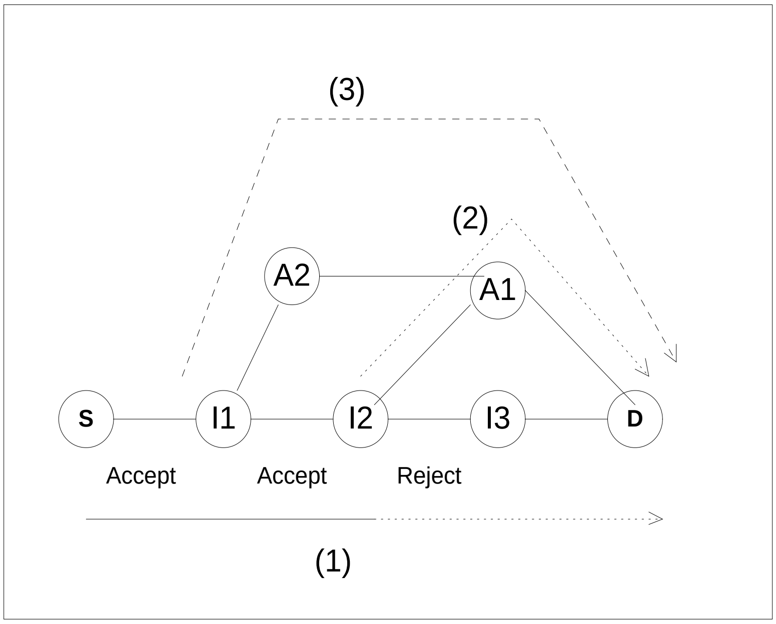

Figure 3.3: Intelligent Restart

Intelligent restarts can also be thought of as a version of retry whereby previously reserved resources are held and a retry is attempted at the node preceding the failed link. This technique should be more beneficial than retries in that retries would return to the source node and utilize routing data that may be out-dated. Also, the route selected may indeed overlap with the previously reserved portion of the last failed establishment attempt[19].

The response time is also dependent on the round trip delay experienced by the establishment message. This round trip delay is the sum of the transmission, propagation, and queuing delays of the packets comprising the message as it travels from the source to the destination node. In this round trip delay only the the packet queuing delays can be affected as the transmission and propagation delays are fixed. The queuing delays can be reduced by reserving resources for the establishment message (i.e., by creating a real-time channel for establishment messages) or by prioritizing the establishment messages so that they are given access to the network before other best-effort traffic. The first method, (i.e., reserving resources for establishment message), which has been attempted in [45], has a significant drawback in that it is very difficult to characterize the establishment traffic passing through a node. We are assuming that a single real-time "management" channel is established between each adjacent node and the management traffic sent over this channel is the aggregate management traffic passing through that node[20]. The second method is more feasible in that the establishment message can be prioritized so

[19]Note that these previously reserved resources would have been released on the return pass after the establishment failed.

[20]It is very expensive to establish a management channel between every node pair in the network.

that it is given preference over best-effort traffic. This method is used in the DCM scheme by inserting a management queue directly below the lowest level real-time queue and above the best-effort queue as shown in Figure 3.4.

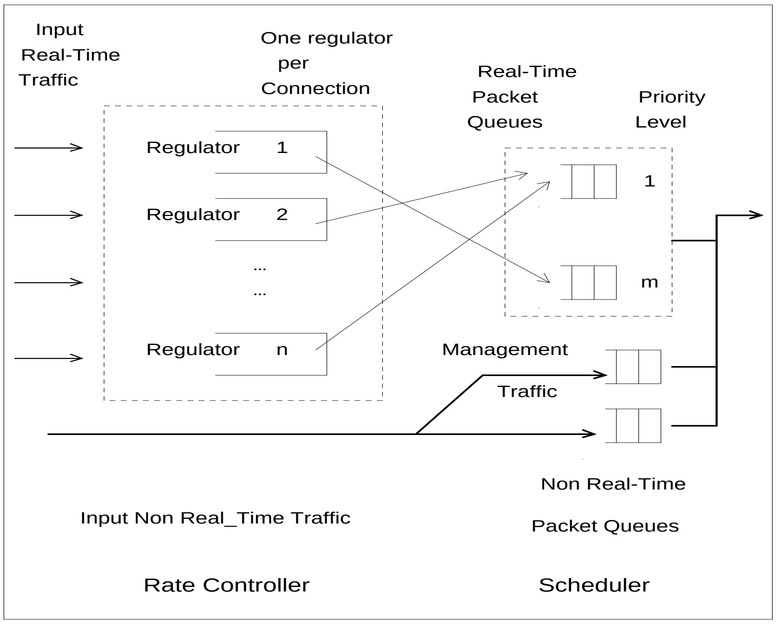

Figure 3.4: Management Traffic in RCSP Queues

The DCM algorithms can be applied to modify the performance parameters or the route of an entire channel. It can also modify a segment of the channel. The smallest segment of a channel that can be modified is a single link. Control can be applied at the link (or *local*) level or at the route (or *global*) level. We make no distinction between a modification affecting a single link and a modification affecting any subset of a channel's links (excluding the entire route); both of these are called *local* modifications. The modification of an entire channel is referred to as a *global* modification. The DCM algorithms can be utilized for both local and global modifications, as the segments of a channel are themselves real-time channels whose source is the node at which the segment begins and destination the node at which the segment ends. The traffic characteristics of these segments are the same as those of the "parent" channel, and the performance requirements are derivatives of those of the parent (i.e., the delay bound is the sum of the delay bounds of the links comprising the segment and the delay jitter bound is the queuing delay in the last switch along the path of the segment). The DCM modification contracts are equally applicable to channel segments.

There are advantages and disadvantages to local and global control. Local control has the advantages that the routes obtained by the routing algorithm usually have a higher success rate than those of global routes, as the routing information gathered within a small radius is normally more accurate, and the channel establishment time is small due to the short distances traveled. However, local routing algorithms do not possess the global knowledge that the source has and may not be aware of routing constraints (these constraints may be cost or security restrictions) and can cause localized saturation as only a small subset of the links are considered.

Transparency Procedures in DCM

The transparency procedure must ensure that the interface to the application client is preserved during the transition from a primary channel to an alternate channel. This transition must be "invisible" to the client as far as this interface is concerned. The interface is usually in the form of a unique identifier that the client uses when transmitting or receiving on the channel[21]. This unique identifier must be preserved at the source and a similar identifier must be preserved at the destination. After the transition, the unique identifier will refer to the alternate channel, and the client's packets will be sent along this alternate route, while at the destination the receiver will continue to receive its packets from its usual receiver abstraction.

While preserving this unique identifier, the client's real-time packets must be switched from the primary to the alternate channel at the appropriate time. The transfer to the alternate channel, while maintaining the same unique source and destination channel identifiers, is done during the alternate channel's establishment. On the forward pass, each intermediate node creates an instance of a channel data structure which stores the state of the channel in that node, and an entry into a virtual circuit routing table that indicates the outgoing virtual circuit identifier and the outgoing link. Along the alternate route, the node preceding the destination configures its virtual circuit routing table so that it points to the same entry as the node preceding the destination in the primary path. In this manner both the primary and alternate channels point to the same destination virtual circuit entry. Note that this action is taken regardless of the conditions of the routes (i.e., whether the alternate and primary routes are completely disjoint, partially disjoint, or identical), as new channel data structures and new table entries are required in all conditions. On the reverse pass the switching between the primary and alternate channels takes place when the virtual circuit table entry corresponding to the source is changed to point to the alternate route. This change is accomplished by modifying the outgoing virtual circuit identifier and the outgoing link identifier at the source node. The previous outgoing virtual circuit and outgoing link identifiers are maintained and used to tear down the primary channel.

An example of this procedure is given in Figure 3.5. The top diagram of this figure indicates the current state of the channel's data structures in the three nodes along the path. Each of these structures contains the local channel identifier (*lcid*), the outgoing link to be used (*link*), and the virtual channel

[21] For example in a Unix based system the unique identifier would be a socket number.

identifier (*vcid*) which is the local channel identifier to be used in the channel structure at the downstream node. In the top diagram the source node uses outgoing link 5 to send its data packets to the intermediate node I1. In addition to a data payload in these packets, there is header information that contains the vcid to be used at this node. At this intermediate node the vcid is used as an index into the table to determine the outgoing link and downstream local channel identifier, which in this case are link 2 and vcid 1, respectively. At the destination node the vcid 1 is used as the index to obtain these link and vcid values. At the destination the outgoing link has a special value that indicates that this packet has arrived at its destination; in this case the vcid indicates the unique identifier for the receiving application. During channel modification the forward pass of the alternate channel establishment message reserves the resources and sets up the data structures in the same manner as that of the primary channel establishment; however, at the destination the value of the vcid is set to that of the primary channel (this keeps the destination interface unique). This can be seen in the second diagram of Figure 3.5, where the data structure[22] indexed by the lcid chosen by that node (lcid 3) has its vcid updated to the unique identifier 92.

On the reverse pass the values of the lcid chosen at each individual node are loaded as the vcids in the upstream node's channels data structure. At the source the unique identifier is maintained by replacing the its current outgoing link and vcid (i.e., that of the primary channel) by the new outgoing link and new vcid (i.e., that of the alternate channel). This is shown in the bottom diagram where the intermediate node, I2, updates its vcid to the lcid of the downstream node (i.e., 3) and the source switches the outgoing link and vcid of the unique identifier 2 (i.e., lcid 2) to 3 and 1, respectively. It should be noted that the lcid originally obtained at the source for the alternate channel, lcid 30, is associated with the original values of the primary channel in order to teardown the primary channel and reclaim the appropriate resources.

3.3 Summary

In this chapter we have presented our solution to the problem of lack of *flexibility* in GPC schemes. *Flexibility* is needed in these GPC schemes in order to adapt to the dynamics of the client's "run-time" demands or to the dynamics of the network state. Our solution, called the Dynamic Connection Management (DCM) scheme, permits the modification of the traffic characteristics, the performance parameters, and the route of channel subject to a modification contract that specifies the extent of disruption to be experienced by the client during this modification. Furthermore, the scheme supports *global* or *local* connection control by permitting modifications to be made to an entire connection or to any of the segments of the connection.

[22]It should be noted that the alternate path passes through a different intermediate node, I2.

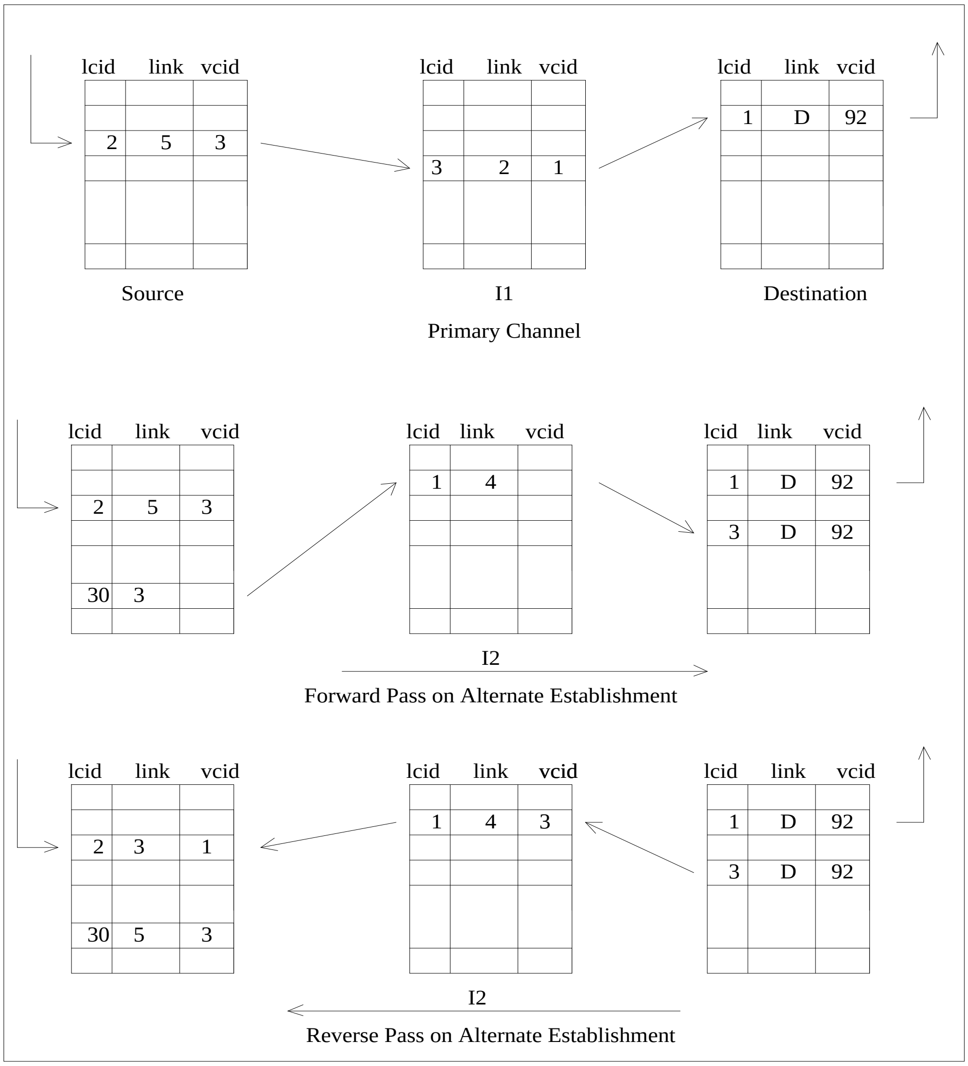

Figure 3.5: Transparency Procedure

The DCM scheme is composed of algorithms and mechanisms. There are three DCM algorithms: the Channel Administration Algorithm, the Routing Algorithm, and the Transition Algorithm. The DCM Channel Administrative Algorithm reserves the network resources, in the presence or absence of resource sharing, needed to support the transition from the original (i.e., *primary*) channel to the new (i.e., *alternate*) channel. This algorithm also ensures that the performance guarantees of the alternate channel are satisfied. The DCM routing algorithm determines a route from the source to the destination host based on the traffic and performance requirements of the connection requested by the client, and the resource sharing factor. The DCM transition algorithm ensures that the performance violations specified in the DCM modification contract are adhered to during the transition. In the chapter we described these algorithms in detail.

The scheme supports mechanisms that enable connection modifications to be made to the segment of the connection (*local control*) or to the entire connection (*global control*), and enables faster establishment and modification of connections. Faster establishment and modification of connections is achieved by the *intelligent restart* establishment procedure and results from the use of the *time value* of the network state information and the ability of the procedure to *navigate* resource saturated links during establishment or modification. Both of these mechanisms were presented and discussed in the chapter.

This chapter provided an in-depth presentation of the DCM scheme. The following two chapters complete this *proof of concept*, in that, Chapter 4 provides an analysis (using simulation experiments) of the algorithms and the mechanism comprising the scheme while, Chapter 5 presents a prototype implementation of the scheme and an analysis of the initial experiments conducted on the prototype.

Chapter 4

Simulation Experiments

In Chapter 3 we provided a detailed description of the DCM scheme consisting of the DCM modification contracts, the DCM algorithms, and the DCM mechanisms. In this chapter we verify and analyze the scheme by a series of simulation experiments the goals of which are :

- to verify that the traffic and performance parameters and the route of an active connection can be modified (both locally and globally), and that the modification contract is honored,
- to verify that the modification times are low (i.e., that the scheme can support the client demand and network state dynamics discussed in Chapter 1),
- to verify that the performance guarantees of all other real-time channels are met, and
- to analyze the performance of the *Intelligent Restart* establishment procedure.

We begin by describing our simulation methodology, and then present the five sets of simulation experiments that were used to examine the DCM scheme, as well as their results and analyses. Some of these simulation experiments are also useful examples of the use of the DCM scheme in a multimedia context.

4.1 Simulation Methodology

In order to accomplish a successful network simulation study, four major issues concerning the methodology must be addressed: the level of detail of the simulation, the network topology, the workloads, and the performance metrics. In this section we explain the methods we used to address each of these issues.

4.1.1 The Simulator

The DCM simulator is an event-driven, continuous-time, C-based simulator that simulates the host (i.e., the source and destination nodes) and switch functionalities at the network layer level. The simulator provides its users with the ability to specify the traffic workload, network topology, routing update algorithm, and DCM policy. The user interface to the simulator is through two files: a command file and a workload file. The command file specifies the topology, the routing update algorithm, the type and quantity of measurement data to be recorded, and various control commands[1]. The workload file specifies a list of connections, with each connection described by its traffic characteristics, performance parameters, beginning and ending times, source and destination hosts, and other per-connection control information[2]. The workload file can also be used to specify error-prone links, together with the probability of an error occurring or the actual times that the errors will occur. The simulator simulates the sending of packets through each of the nodes (i.e., hosts or switches). These packets can be real-time data packets, best effort data packets, routing update packets, or establishment (both initial and modification) packets. In the simulation, packets are queued in the appropriate RCSP queues (each queue corresponds to a distinct priority level) and forwarded at the appropriate times; also propagation and transmission delays are accounted for at each of the links. Establishment and modification tests are conducted at each node along the path, and routing is done at the source (or, in the case of intelligent restart, at other nodes along the path). Routing update messages propagating throughout the network are simulated when connections are established or modified[3]. Data is gathered during the course of the simulation experiment based on the variables that are specified in the command and workload files. The metrics derived from this data is discussed in Section 4.1.4.

4.1.2 Network Topology

As we are investigating a GPC scheme, it would be advantageous to conduct simulation experiments using the topology of a network on which GPC services will be provided, such as XUNET[24]. However, as these types of experimental networks are in the initial stages of their development, their topologies are somewhat sparse and would not provide a suitable model on which to do a useful analysis. In order to facilitate useful analysis, we chose a square mesh topology that allows us to analyze a complex network (as there are multiple paths through the network) while permitting quick insights from and verifications of the results (due to the symmetry of the topology). The topology of the simulated network and its configuration are given in Fig. 4.1. This square mesh consists of 25 nodes and 40 links; each one of these

[1]These control commands select the type of establishment procedure to be used and the format of the reports that are to be generated.

[2]This control information specifies the trace level to be used on the connection for debugging purposes.

[3]Periodic updates are also used if there are infrequent establishments and modifications.

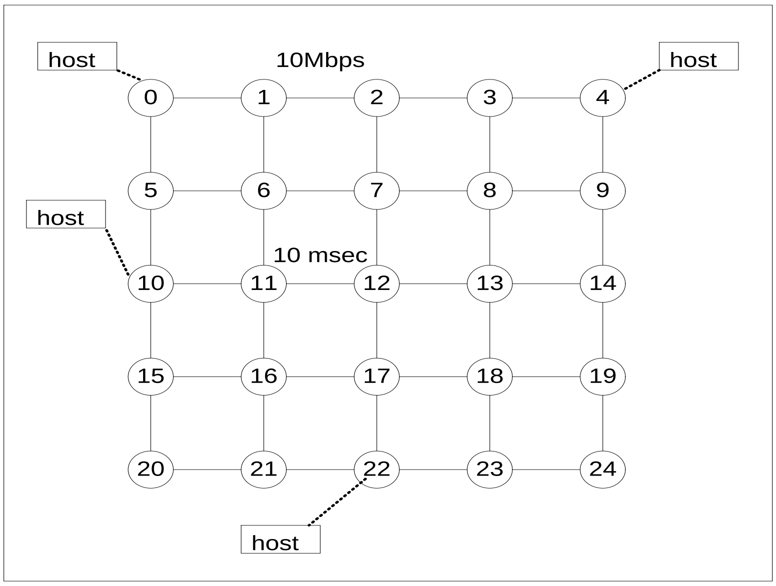

Figure 4.1: Experimental Network

links represents a pair of simplex links with opposite directions. All of the links in this network have a speed of 10 Mbps and a propagation delay of 10 ms. Attached to each of the nodes on the periphery is one host which is connected by a local link. The hosts are assumed to be attached to the nodes by very fast LANs (e.g., FDDI), hence their link speeds are not bottlenecks, and are not considered during the simulation experiments. In Fig. 4.1, four of the sixteen hosts are shown, hosts 0, 4, 10, and 22.

4.1.3 Workload

The workloads used in these simulation experiments are composed of the set of real-time channels that are established, modified, and terminated during the course of the experiment, with a background load comprised of real-time and non-real time connections and best-effort management traffic. To study its performance, the DCM scheme was examined over a variety of workload levels. The determination of these workload levels is difficult as it depends on the number of channels, their traffic and performance specifications, and the length of their routes. It would be useful if this workload level could be represented by a single-valued metric that encompassed all of these factors, as it would allow us to easily compare any two workloads (or any two channels). To put this in context, how could two workloads be compared where 15 audio (low bandwidth/low delay) channels and 2 video (high bandwidth/medium delay) channels are established in the first workload and, under identical pre-conditions, 4 video channels and 9 audio channels are established in the second workload? A first attempt at a solution to this problem might be to use the bandwidths of the channels as the only criterion. The problem with this approach is that it ignores path-length and delay. If two channels have the same traffic and performance characteristics but different path lengths, the channel with the longer path uses more resources within the network, hence it should have a greater effect on the load in the network. Delay is especially important in guaranteed service networks since giving a low delay to a channel involves giving it higher priority service and effectively blocking more future channels. Thus, a low delay request uses more resources within the network, as we will see later on in this section.

A general solution to this problem is quite complex, and, in fact, would solve more than the relatively academic problem of comparing workloads or channels. For instance, such an index could be used for pricing, as a resource allocation tool, or as a routing index in guaranteed service networks. For the needs of our simulations, we have devised a simple index which meets the above requirements within the context of the RCSP scheduling discipline.

The description of the RCSP admission control scheme, in section 3.1.3, motivates the choice of the load index in a network using this scheduling strategy at all link. The RCSP admission control bandwidth test ensures that, at any priority level p, when a packet arrives, the maximum amount of time it could possibly wait before it can be sent out is bounded by the delay bound $\overline{d^p}$ corresponding to that level. This bound must take into account any packets at the same level which could be there before the packet arrived, and any packets which could come in at any higher level before the packet is sent out. Thus, any time a channel is added to level k at a link, we have to ensure that, for any level $p \geq k$ (i.e., equal or lower priority levels), the amount of work (in terms of transmission time of the packet) which can come in during a time period $\overline{d^p}$ is less than $\overline{d^p}$. In other words, adding a channel at level k adds a certain amount of *work* to all equal and lower priority levels $p \geq k$. The admission control tests ensure that the total *work* at any level p does not exceed the delay bound $\overline{d^p}$ for that level, where *work* at level p is defined as

$$W_p = \sum_{j=1}^{C_p} \lceil \frac{\overline{d^p}}{Xmin_j} \rceil * \frac{Smax_j}{l} + \frac{\overline{Smax}}{l} \tag{4.1}$$

where C_p is the number of connections at level p, $Xmin_j$ is the minimum inter-arrival time of the packets on the jth connection, $Smax_j$ is the maximum size of the packets on the jth connection, $\overline{Smax}$ is the size of the largest packet that can be transmitted on the link, and l is the link speed. Introducing a channel into the network at level p adds *work* to this level and all lower levels. This is why a low delay (high priority) channel consumes more resources in the network, since it uses a high-priority level (small p) and it increases call blocking probability at all levels $m > p$. If the delay requirement is larger, it goes into a lower priority level (larger p) and affects fewer levels. This prompts us to consider the sum of the *work* across all levels as an index of the load on the link. A high priority channel will cause a big change in this index as it will increase the *work* at all levels, while a channel at the lowest priority level will only affect one level. By summing this index of the load across all of the links and dividing by the number of levels in the RCSP server[4] we obtain an index of the load on the network. We call this index of the load on the network the **Queuing Delay Index**. For the sake of brevity, in the remainder of this chapter we will refer to this index as the *Load Index*.

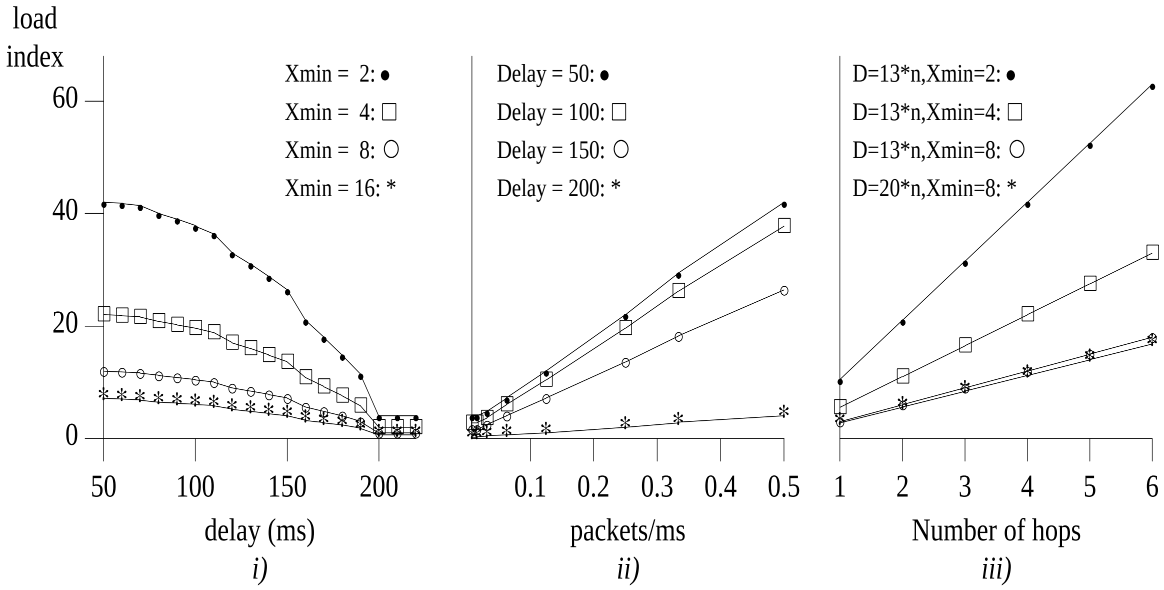

Figure 4.2: Queuing Delay Index

Figure 4.2 shows how successful this index is at capturing bandwidth, delay and path-length as components in the overall load on the network. Graph *i)* shows how the index changes as a function of the delay requirement of a channel on an otherwise empty network. We can see that as the delay requirement becomes slacker (higher end-to-end delay bounds) the load index decreases. The graph flattens out beyond 200ms because at this point the channel is at the lowest level in the RCSP queues and further increases in the delay bound do not affect the priority level of the channel. Thus, bandwidth and path length remaining the same, the index is monotonically decreasing as a function of the end-to-end delay bound. Graph *ii)* shows that the load index is linear in the bandwidth of the traffic, with the slope of the line dependent on the delay bound of the channel. Graph *iii)* show that the index increases linearly with the path length, as long as the delay bound per link and the $Xmin$ value of the connection remain constant. We believe that this index is a good characterization of the overall load on the network. We will use this index to characterize the load on the network in order to study how the behavior of the scheme changes as a function of the network load. This index also makes it possible to compare channels which have different bandwidth, delay and path lengths. To better examine the performance of the DCM scheme, the simulation experiments were conducted over a variety of workloads. For the sake of analysis the workloads were grouped into three discrete workload levels: *low*, *medium*, and

[4]We assume that all RCSP servers in the network contain the same number of priority levels.

Workload level	Load Index Range	Average Bandwidth Utilization	Maximum Bandwidth Utilization
low	244 - 294	16.52%	52.6%
medium	697 - 747	40.32%	78.40%
high	1208 - 1268	64.17%	89.96%

Table 4.1: Workload Levels

high. These workload levels correspond to a range of load indices. It should be re-emphasized that the *load index* is a composite of the bandwidth and delay resources, hence a load index can represent a variety of combinations of bandwidth and delay resources. In order to give the reader some idea of the bandwidth resource reserved at a specific load index, one of the possible bandwidth utilization values corresponding to the maximum load index of the workload level is presented in Table 4.1 for each of the three workload levels. In Table 4.1, *Average Bandwidth Utilization* refers to the average bandwidth reserved across all of the links in the network (this bandwidth is represented as a percentage of the link speed), while the *Maximum Bandwidth Utilization* refers to the bandwidth reserved in the heaviest loaded link in the network (this bandwidth is again represented as a percentage of the link speed). As the network load index captures more than the bandwidth information, a set of channels with identical bandwidth requirements but different delay requirements or path lengths would give us very different load index values. Also, setting up the same load index values with the same statistical distribution of channels using different seeds in the random number generator would give us slightly different values of network utilization. The ranges used for each level were chosen by determining the minimum and maximum possible values of the load index for this topology and channel mix, and then deriving three smaller ranges from this broad range. This derivation is heuristic and entirely for convenience in these experiments, and is only pertinent to these simulations (i.e., this topology, with these specific connection types and with this mix). The maximum possible value was determined using the workload mix described below, and is actually an average of the load indices obtained using many different seeds to generate the workload sets. Three sub-ranges were then selected to define each workload level.

The workloads themselves were statistically generated with the following characteristics. The channels in the workload come from three classes, with parameters corresponding to

- A - a one-way low-quality video conference channel,
- B - a CD-quality audio channel, and
- C - a telephone-quality audio channel.

Channel Class	Xmin *(ms)*	Xave *(ms)*	I *(ms)*	Smax *(bits)*
A	5.0	5.0	500.0	10000
B	50.0	50.0	500.0	10000
C	153.0	153.0	500.0	10000

Table 4.2: Channel Classes

The traffic and performance parameters of each of these channels classes are given in Table 4.2. The distribution of these classes is chosen so that 30% of the channels belong to class A, 30% to class B, the the remaining 40% to class C. An analysis of several different distributions of these classes showed minor differences in the resulting values of various performance metrics[5]. The host pairs were selected randomly to lie along the periphery of the mesh. The delay requirements of the channels were also generated statistically, to lie uniformly in the range $[x + 50, x + 100]$ ms, where x is the one-way propagation delay between the source and the destination of the channel. In the experiments where a jitter bound value is needed, the jitter bound values were chosen to lie uniformly in the range $[1, 40]$ ms[6]. Thereafter, different numbers of channels were generated with the above statistical properties to get the appropriate values of the load index corresponding to the desired workload level.

In all of the experiments there was at least one real-time channel originating from each host and all links had real-time traffic present. During the course of channel modifications in each of the simulation experiments new channels were being established and existing channels were terminated. The channels generated to create these workloads constituted the background loads. One or more of these channels were then randomly selected (or tagged) as the channels to be modified[7], and the experiments were conducted on these tagged channels. There was a best-effort load present during all simulations; however, this load was entirely composed of channel management packets (i.e., packets from establishment, modification, and termination messages) and routing update packets.

4.1.4 Performance Metrics

The performance metrics of interest in the experiments are the throughput[8] of the channels, the end-to-end delay and delay jitter experienced by the packets as they traverse the route, the numbers of out-of-sequence packets received by the destination hosts, and the channel establishment and modification times. This data was collected on all channels, that is, channels that were modified using the DCM scheme and on "background" channels that were not modified. The data collected on the unmodified

[5]These metrics are the discussed in Section 4.1.4.

[6]The lower and upper bound of this distribution are based on the minimum and maximum delay values that we believe can be easily supported in a switch.

[7]In these experiments at most three channels were modified simultaneously.

[8]For simplicity, in the simulations we assumed that all of the bits sent in a packet were useful.

channels is used to ensure that the performance guarantees provided to these "background" channels are met in the presence of channel modification.

A count of the number of packets traversing each node in a real-time channel was recorded and used to determine the number of packets arriving at the destination node. This packet count at the destination node, together with the packet size, is used to compute the throughput of the channel at the destination host over the channel's lifetime. This throughput is displayed in a graph of bits per second versus simulation time. The end-to-end delays of all packets traversing each real-time channel are recorded and displayed in a delay histogram. This histogram is used to determine the number of packets exceeding their delay bounds and the delay jitter experienced by packets traversing the connection; we can thus detect jitter bound violations. An *out-of-sequence* counter is kept at the destination node of each real-time channel and used to record all out-of-sequence packets that are passed to the destination application. All simulation data was taken with a millisecond granularity, and the simulations were run for one to two hours of simulated time (i.e. 3.6×10^6 ms to 7.2×10^6 ms) depending on the simulation experiment.

The establishment times of the connections (both initial establishments and modifications) are important in assessing the viability of DCM. The establishment time is the sum of the communication delays (i.e., propagation, transmission, and queuing delays), and the processing times of the establishment message (not including the admissions test), the execution of the routing algorithm (at the source node only), and the execution of the admission control tests. Since the communications delays and, to some extent, the message processing times are beyond our control, we shall focus on the processing times of the admissions tests and of the routing algorithm. The admission control tests are executed on a per-link basis for each link traversed by the establishment message along the path. As can be determined from the admission control tests (Sections 3.1.3 and 3.2.2), in the worst case the order of growth for this algorithm is $O(PM)$, where P is the number of links in the path and M is the number of priority levels in the RCSP queue. Under worst-case conditions, the routing algorithm must be executable at each node along the path. The algorithm first forms the directed graph, and then runs the constrained-modified Bellman Ford algorithm (described in Section 3.2.2). Thus, in the worst case, the order of growth for this algorithm is $O(QNL + QPM)$, where N is the number of nodes in the network, L is the number of links in the network, Q is the number of nodes along the path, and P and M are as described previously. Using the dimensions of the experimental network (there were ten priority levels in the RCSP queues) and running the algorithms on a DECstation 5000/240, the admission control tests took 3.3 milliseconds at each node. In the worst-case scenario the routing algorithm took 5.5 ms on the same machine. It should be noted that the order of the routing algorithm's runtime assumes worst-case conditions (i.e., in which the network is very heavily loaded, and at each node encountered by the establishment message the initially chosen link is unavailable). These values are used in the simulation experiments in order to obtain worst-case establishment times. We also fix the message processing times at each node (based on our initial experience with the Real-time Channel Admission Protocol (RCAP) of the Tenet protocol suite) as 2.5 ms [7]. The establishment times are recorded for each real-time channel in the network.

4.2 Experiments

4.2.1 Overview of Experiments

The overall goal of the simulation experiments is to verify that active connections can be dynamically modified in conformance with the modification contracts and without violating the performance guarantees made to other real-time connections. Thus the sub-goals are :

- to verify that the traffic characteristics, performance parameters, and the route of an active connection can be modified (both locally and globally),
- to verify that the modification contract is honored during their modifications,
- to verify that the modification times are low (i.e., that they can support the client demand and network state dynamics of interest to us),
- to verify that the performance guarantees of all other real-time channels are met, and
- to analyze the *Intelligent Restart* establishment procedure.

The performance metrics mentioned in the previous section were determined for a suite of five experiments designed to achieve all of the sub-goals stated above. The first four experiments were designed in a manner that depicted the usefulness of the DCM scheme in supporting multimedia connections. The scheme can be used for supporting the guaranteed performance needs of any type of application; the multimedia simulation examples provided here are simply used to place the scheme in a familiar context. The first three experiments focus on client dynamics, and are mainly associated with the modification of the traffic characteristics and the performance parameters of a client's channel. These experiments illustrate client-initiated modifications. The fourth experiment is a network-initiated modification and depicts a situation where the dynamics of the network state require the transparent modification of the route of a channel. The final experiment examines the *Intelligent Restart* establishment procedure to determine its usefulness and cost.

The first experiment is that of a *still image lossless browser* which is used to browse a sequence of large still images. These images are played back at low speeds, and cannot tolerate lossy compression; hence, browsing is best achieved by enhancing the bandwidth of the existing real-time channel by a factor greater than 1. Usually this enhanced bandwidth is maintained for a short duration, after which the application requests a reduction of the channel's bandwidth to its original value. In this example it was initially necessary to tune the performance parameters, as insufficient bandwidth was requested. This situation can arise when the client has insufficient traffic information and chooses traffic parameters that provide unacceptably poor image quality.

The second experiment is an example of client dynamics associated with the quality of the end-user result. The example presented here occurs when the application selects delay values that are insufficient to meet the human user's needs. This situation can arise when a client has incorrect or insufficient data concerning its performance requirements. In this experiment, the delay bound is adjusted to provide a comfortable level of service. The delay bound of the channel is modified under a Bounded-Violation modification contract. This experiment differs from the previous experiment in that all modifications in that experiment are subject to a No-Violation modification contract.

The third experiment illustrates another aspect of client dynamics associated with the quality of the end-user result. In this example the jitter bound provided on the stream is very large and the destination application does not have enough buffer resources to support this jitter. In this experiment the jitter bound is adjusted to provide a more stringent bound. This jitter bound is modified under a No-Violation modification contract.

The fourth experiment focuses on the route modification aspect of the DCM scheme, and involves the dynamics of the network state. This experiment examines the responsiveness of the scheme in the presence of error prone links in the network. In this scenario, a channel specifies an error threshold and the network monitors the links to determine when this threshold has been exceeded. The error threshold is a limit on the number of consecutive packets containing bit errors[9]. Rerouting can be global or local; however, only local rerouting is used in this experiment, which also illustrates the ability of the network to redistribute its load in the presence of reclaimed network resources[10] and to increase its availability to new or current connections. In this experiment, no performance violations must be experienced by the modified channels, as the modifications must be *transparent* to the client.

It should be noted that in the first three experiments route changes are possible for two reasons: 1) the primary route of a connection may not have sufficient resources to allow a performance increase, and 2) the routing algorithm may select a different route so as to better *balance* the network's load.

The last experiment examines the *Intelligent Restart* establishment procedure, comparing it to the use of establishment retries. The metrics used in this comparison are the total establishment (or modification) time and the load index of the channel. The load index of the channel is the Queuing

[9]Fault recovery is also possible using this mechanism, as local rerouting can be used to reroute that portion of the network that is faulty while maintaining resources previously acquired in the fault-free section.

[10]Resources can be reclaimed for maintenance or administrative purposes.

Parameters	Initial	Playback	Fast Browse
Xmin *(ms)*	6.0	5.0	2.5
Xave *(ms)*	6.0	5.0	2.5
I *(ms)*	500	500	500
Smax *(bits)*	10000	10000	10000
$\overline{D}$ *(ms)*	80	80	80
Bandwidth *(Mbps)*	1.7	2.0	4.0

Table 4.3: Traffic and Performance Parameters for Browser Experiment

Delay Index of a network containing only this connection, which provides a measure of the resources consumed when using each procedure. This last experiment is conducted across all workload levels in order to determine the effect of the workload on the establishment procedure.

In all of these five experiment sets the DCM scheme should ensure the following that:

- the delay and delay jitter bounds of all packets on the primary and alternate channels are met;
- the throughput of the alternate channel correctly reflects the modification (if a bandwidth modification has been made);
- there are no out-of-sequence packets or, in the case of a bounded-violation contract, the number of out-of-sequence packets does not exceed the violation bound; and
- there are no performance violations on any other real-time channel.

4.2.2 Experiment I - Image Browser

In the first experiment, Experiment I, the application considered is that of a *still image lossless browser*, which is used to browse a sequence of large still images. This type of application is used, for example, by environmental scientists who examine large satellite maps to determine minute changes over a certain period of time. Lossy compression techniques cannot be used on these images as they may remove these minute changes, and low frames speeds are used in normal playback operation. As these images cannot tolerate lossy compressions, faster browsing is best achieved by increasing the bandwidth of the channel by any factor that permits useful work by the user. The traffic and performance parameters of the different states of the connection are provided in Table 4.3 below.

In this experiment the source of the connection is host 10, the destination is host 14, and the initial route chosen traverses the switches 10, 11, 12 ,13, and 14 (shown in Figure 4.3 as (1)). The network load during these modifications was at the *high* workload level and had an average load index value of 1209.4.

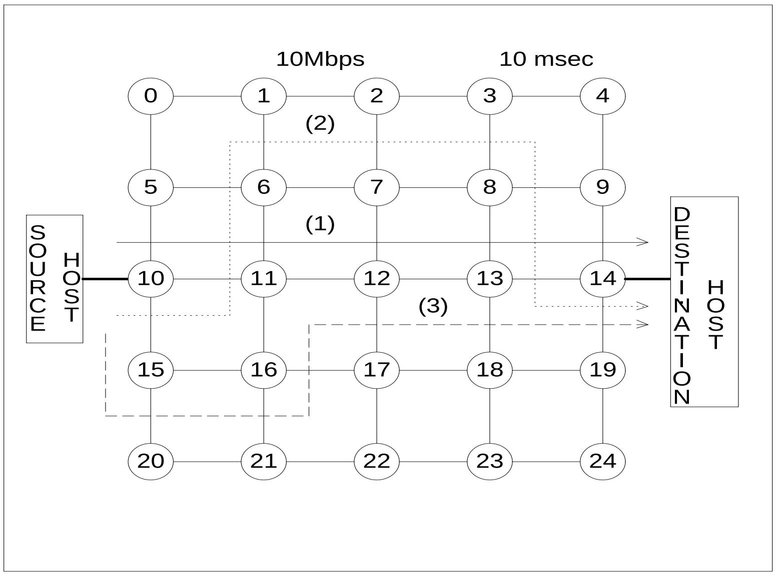

Figure 4.3: Experiment I - Browser

Initially, insufficient bandwidth was requested for the channel, and the traffic parameters had to be tuned to obtain adequate bandwidth. To this end, an initial request was made for a throughput performance increase of 15.0%. After tuning the parameters of the channel, channel modifications were done to reflect the application's needs for fast browsing and playback. The parameters associated with each of these states, fast browsing and playback, are also provided in Table 4.3. The bandwidth of the channel is doubled for the duration of time that the client needs the higher speed (this is usually short compared to the channel's lifetime), after which the application requests that the bandwidth be reduced to its original value. In this experiment the start of the fast browsing period and its subsequent duration were statistically derived. The starting times of the browsing periods were randomly chosen from an exponential distribution with mean 10 minutes, and the durations randomly chosen from a uniform distribution with an upper bound of 15 minutes and a lower bound of 5 minutes. The network's real-time background load was dynamic in that real-time channels were being created and terminated during the entire experiment, thus encouraging route changes to accommodate alternate channels. There were three route changes (illustrated as (1), (2), and (3) in Figure 4.3[11]), which exercised the routing algorithms under the resource sharing constraints. The results of this experiment are displayed in Table 4.4 and Fig. 4.4.

[11]Route (1) traverses the nodes (10,11,12,13,14), Route (2) traverses the nodes (10,11,6,7,8,13,14), Route (3) traverses the node (10,15,16,11,12,13,14).

Metrics	Values
Number of Modifications	9
Number of Route Changes	4
Maximum Setup Time *(ms)*	162
Average Setup Time *(ms)*	133.5
Minimum Setup Time *(ms)*	108
Maximum Packet Delay *(ms)*	79
Average Packet Delay *(ms)*	62.4

Table 4.4: Results of Browser Experiment

In the experiment there were 9 channel modifications (i.e., one channel tuning and 4 browsing periods) with three route changes. The average (round trip) and maximum modification times, 133.5 and 162 ms respectively, reflect the changes in routes due to modifications. The average and maximum packet delays indicate these changes as well, and also verify that no packet exceeded its delay bound of 80 ms, as can be seen in Figure 4.5[12]. No out-of-sequence packets were observed. The throughput and delay bounds on all of the other channels were met throughout this experiment. This experiment verified the DCM scheme in that modifications were accomplished within the constraint of the modification contracts.

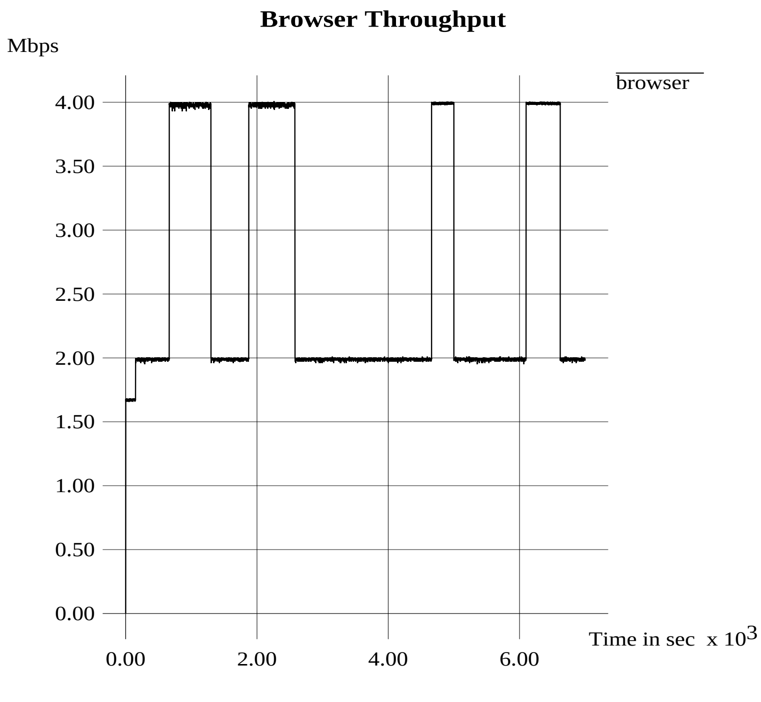

Figure 4.4: Throughput of Browser

[12]As mentioned in Section 3.1, the RCSP delay jitter servers are non work-conserving servers and hold packets until their eligible times; therefore, there is both a minimum and maximum delay bound associated with the channel. This is shown in the histogram in Figure 4.5, where the lower bound on this connection is 51 ms

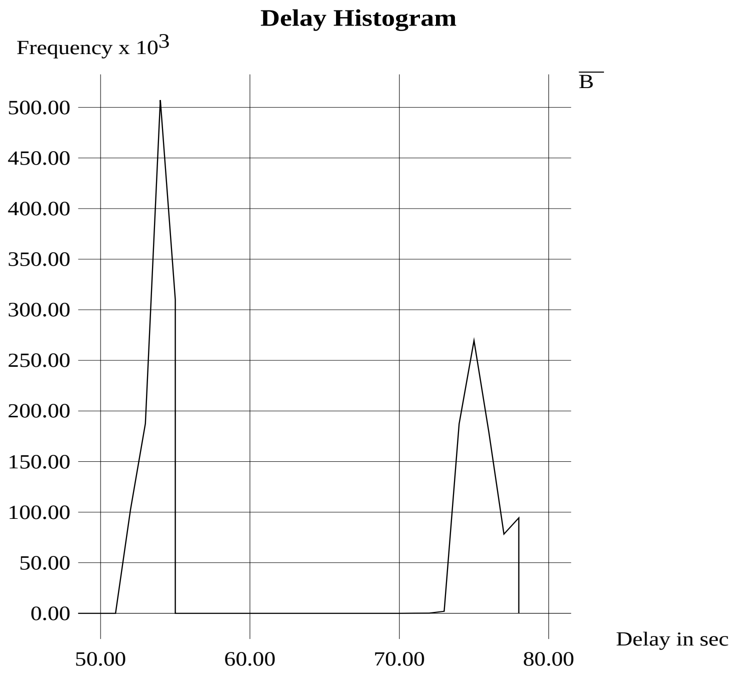

Figure 4.5: Delay Histogram of Browser

It should be noted that the maximum alternate channel modification time along this route is 162 ms, of which 120 ms was propagation delay; this value is less than the values associated with electro-mechanical switches such as those found on VCRs (the time between activating the switch and the response is usually in the range 0.5-0.8 sec). The maximum alternate modification time is dominated by the propagation delay, which accounts for 72.9% of the total.

4.2.3 Experiment II - Delay Bound Modification

The delay experienced by a real-time channel between hosts 10 and 14 was excessive, and a channel modification request was made to the network to reduce the delay bound from 160 ms to 90 ms to 60 ms. The topology of the network was as shown in Figure 4.1. The traffic and performance characteristics, and the route of the original and adjusted channels are given in Table 4.5. The network loads during modifications were at the *medium* workload level, with an average value of 724.5.

The route changes in this experiment resulted in some out-of-sequence packets. As the delay bound conditions given in Eq. (3.7) cannot be met[13], a "bounded-violation" modification contract is used for the modification. This modification contract stated to the client that the maximum number of packets that could violate their performance bounds during *Adjustment 1* and *2* (see Figure 4.5) are 23 and

[13]For simplicity, in this experiment the local delay bounds were *relaxed* so that $\overline{D^p}_{path} = \overline{D^p}$ and $\overline{D^a}_{path} = \overline{D^a}$ in both the **initial** and **Adjustment 1** case.

Parameters	Initial	Adjustment 1	Adjustment 2
Xmin *(ms)*	5.0	5.0	5.0
Xave *(ms)*	5.0	5.0	5.0
I *(ms)*	500	500	500
Smax *(bits)*	10000	10000	10000
$\overline{D}$ *(ms)*	160	90	60
Route	10,5,0,1,2,3,4,9,14	10,5,6,7,8,9,14	10,11,12,13,14

Table 4.5: Traffic and Performance Parameters for Experiment II

$\overline{D_p}$	$\overline{D_a}$	*Bound*	*Final Bound*	*Packet Count*
160	90	23	14	13
90	60	9	6	5

Table 4.6: Result of Experiment II

9, respectively. Table 4.6 provides the packet violation bounds and the actual number of packets that violated their performance bounds. The establishment time was 218 ms, and the modification times were 128 ms and 114 ms, respectively. These delays are short given that the filtering capability of human vision is incapable of distinguishing any pauses between frames 33 to 70 ms apart [36]. Hence the visual response seems reasonable, especially when the propagation delay is on average 100 ms. In LAN and MAN environments the delays should be well within the distinguishing range mentioned above.

In Table 4.6, $\overline{D^a}$ is the modified delay bound requested by the client, $\overline{D^p}$ is the delay bound along the primary route, *Bound* ($\lceil \frac{\overline{D^p}_{path} - Xmin^a - \overline{D^{min}}_{path}}{Xmin^a} \rceil$) is the offered performance violation bound, *Final Bound* is the tighter bound as given by Eq. (3.8), and *Packet Count* is the number of packets whose performance contracts were actually violated. As can be seen in Table 4.6, the DCM performance-violation guarantees were met.

4.2.4 Experiment III - Jitter modification

In this experiment the jitter bound provided to an application was incorrectly chosen, and the resulting buffer usage at the application could not be supported. As a result the client requested a decrease in the delay jitter bound of the channel. As discussed previously, the delay jitter bound of a real-time channel is the local delay bound in the destination node ($\overline{d_n}$); hence, a decrease in the jitter bound corresponds to a decrease in the delay bound at this node. As this local manipulation must be done at the destination node and the request is client-initiated, the establishment message must traverse the entire route.

The channel modification request was made to the network to reduce the jitter bound from 8 ms to 4 ms. The traffic and performance characteristics of the original and adjusted channels are given in Table 4.7; the network load during modification was at the *medium* workload level with a value of 727. The

Parameters	Initial	Adjustment 1
Xmin *(ms)*	5.0	5.0
Xave *(ms)*	5.0	5.0
I *(ms)*	500	500
Smax *(bits)*	10000	10000
$\overline{D}$ *(ms)*	80	80
$\overline{J}$ *(ms)*	8	4

Table 4.7: Traffic and Performance Parameters for Experiment III

topology of the network is as shown in Figure 4.1, with the channel's source host was at node 0, and the destination host at node 4. In this experiment no route changes occurred. The establishment and modification times were 112 ms and 118 ms, respectively. The delay histograms, Figure 4.6 and Figure 4.7, depict the delay distributions before and after the jitter bound modification.

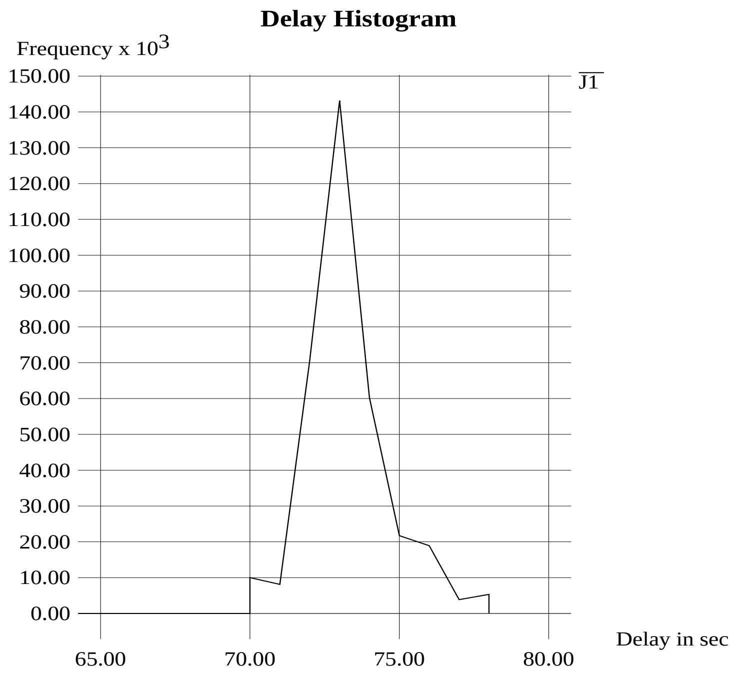

Figure 4.6: Experiment III - Jitter Bound of 8 ms

As can be seen from these diagrams, the jitter bounds before and after the modifications are met: indeed, the spread of the delay distribution in Figure 4.6 is 8 ms, and that in Figure 4.7 is 4 ms. It should be noted that the RCSP servers in the DCM scheme are delay-jitter controllers; thus, packets are held in each node until their eligible time. This creates a lower and an upper bound on the end-to-end delay of packets; the lower bound in this experiment is 70 ms, and the upper bound is the sum of the lower bound and the maximum possible delay jitter. The throughput diagram, Figure 4.8, also indicates that the changes in the jitter have no effect on the throughput of the channel. This modification was

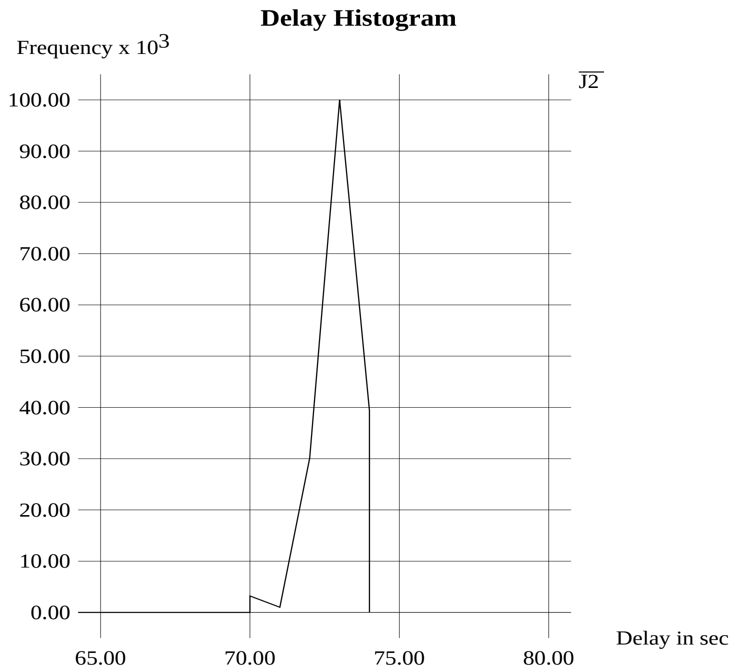

Figure 4.7: Experiment III - Jitter Bound of 4 ms

done under a No-Violation modification contract, and there were no performance violations on this or any other real-time channel.

4.2.5 Experiment IV - Error Management

In this experiment real-time channels have error thresholds associated with them, and the network monitors errors on all links to determine whether the error threshold of a channel is exceeded. If the threshold of a channel has been exceeded, the network then attempts to reroute the channel (however, it does not discard corrupted or corrupted packets). Link errors are generated uniformly on all links, with a mean time between faults of 10 minutes. In this experiment one such channel is considered, with Table 4.8 indicating the faulty links and the new and old route used by the connection. The faulty link is denoted by a pair *(s,d)*, where *s* is the source node of the link and *d* the destination node. The connection had the following parameters: $Xmin$ = 5.0 ms, $Xave$ = 5.0 ms, I = 500, and D = 110 ms. The source host of the channel is at node 0, and the destination at node 4; the new routes are illustrated in Figure 4.9. The network load during these modifications was at the *medium* workload level, with an average load index value of 741.6.

Initially the error threshold was exceeded on link (1,2) (i.e., threshold (1) indicated in Table 4.8), and the source node of the faulty link, node 1, attempted a **local** reroute of the channel and succeeded in rerouting that portion of the connection on link (1,2) to links (1,6), (6,7), and (7,2). The other two

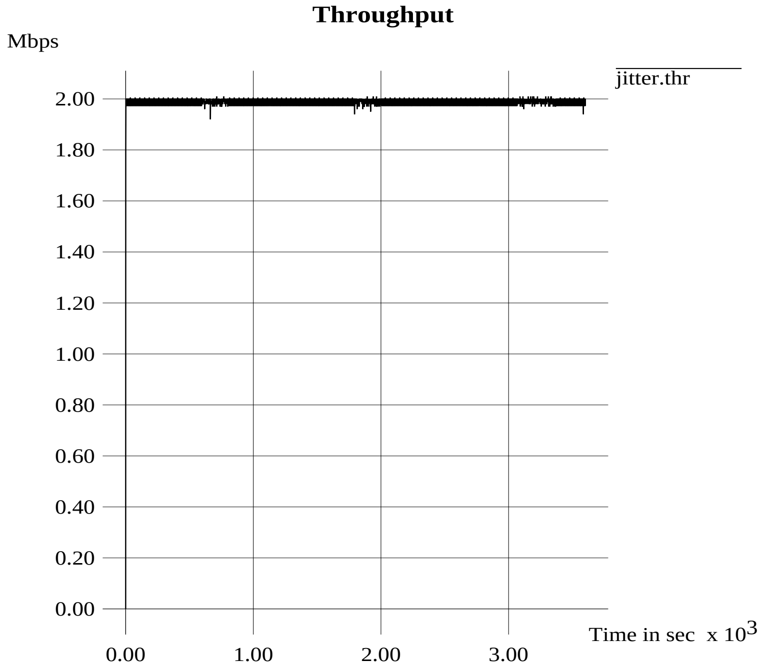

Figure 4.8: Experiment III - Jitter Modification

Threshold	**Faulty Link**	**Current Route**	**New Route**	**Local Modification Time**
1	(1,2)	0,**1**,**2**,3,4	0,**1**,**6**,**7**,**2**,3,4	86 ms
2	(7,2)	0,**1**,**6**,**7**,**2**,3,4	0,**1**,**6**,**7**,**8**,3,4	62 ms
3	(0,1)	0,**1**,**6**,**7**,**8**,3,4	0,**5**,**10**,**11**,**12**,**13**,**8**,3,4	169 ms

Table 4.8: Fault Status

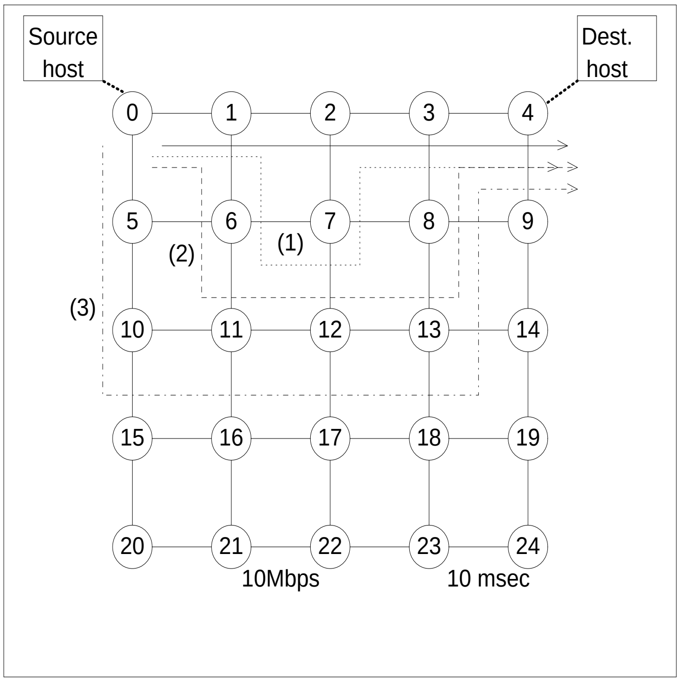

Figure 4.9: Error Recovery

occasions in which the thresholds were exceeded occurred on links (7,2) and (0,1), and resulted in the local reroutings highlighted in bold in Table 4.8. The local modification times are given in Table 4.8. On average 69.2% of the modification times are due to propagation delays. The throughput diagram (Figure 4.10) shows no reduction in throughput as the connection is rerouted, since no packets are lost during the reroute. It should be noted that corrupted packets are not discarded but transmitted at the destination to upper-layer error recovery mechanisms. The histogram (shown in Figure 4.11) indicates the three route changes as they cause three distinct distribution segments centered at 44, 68, and 86 ms, respectively. It should also be noted that no packet exceeds its delay bound of 110 ms, and that there were no out-of-sequence packets. This rerouting is transparent to the client, as all performance guarantees are met (i.e., there are no guarantee violations). The performance guarantees of all other channels in the network were also met.

The experiment also illustrates the ability of the network to use local rerouting to transparently redistribute the load on the network, so that a high performance (i.e., resource intensive) channel can be admitted into the network. This increases the availability of the network to high performance multimedia applications. In this scenario, the admittance of a channel into the network could only be permitted by the rerouting of a portion of a current channel. This situation frequently occurs due to the dynamic nature of client demands (i.e., the channel origination times and the channels' lifetimes) as connections are created and terminated.

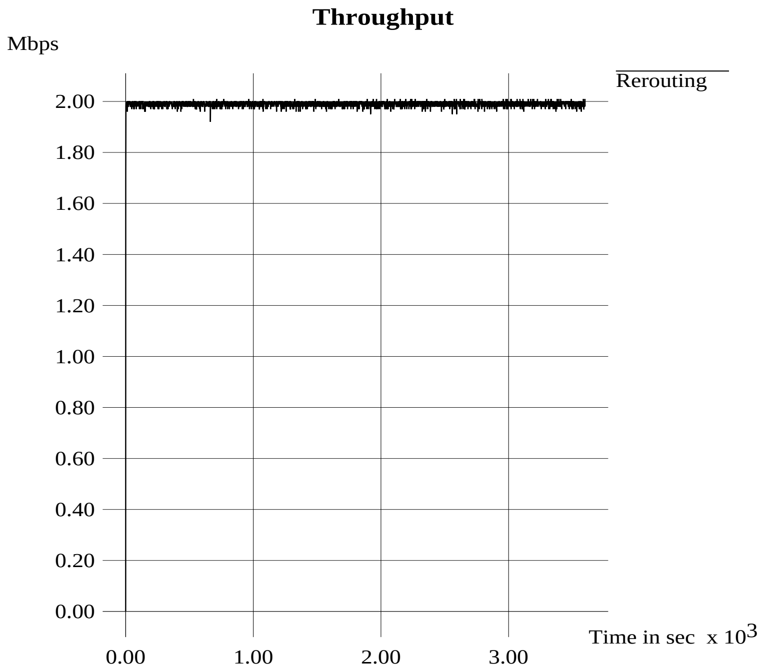

Figure 4.10: Error Recovery - Throughput Graph

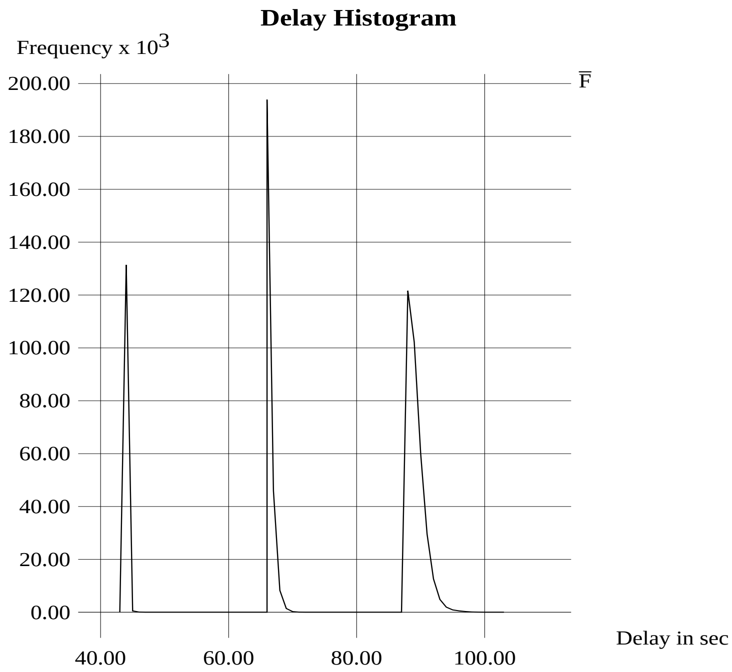

Figure 4.11: Error Recovery - Histogram

4.2.6 Experiment V - Intelligent Restart

In this experiment we do an initial examination of the *Intelligent Restart* establishment procedure by comparing it with the conventional method of *retries*. In the conventional method, if a channel establishment[14] is unsuccessful, the source node initiates other establishment attempts until it succeeds in establishing the channel or it exceeds some retry threshold (i.e., the maximum number of retries permitted). The time between consecutive establishment attempts and the value of the retry threshold are variables that are set by the DCM policy. As discussed previously, the *Intelligent Restart* procedure is a version of retries in which the retry begins at the node preceding the unavailable link (i.e., the link on which the admission control test failed) and attempts to find a route from this node to the destination host that (in conjunction with the previously reserved resources) satisfies the traffic characteristics and the performance parameters requested by the client. In these experiments the time between consecutive attempts is zero, that is, the retry attempt is started immediately. However, the routing algorithm updates its database to reflect the unavailable link, and then determines a route from the source to the destination host. We will not examine the impact of this factor (i.e., we will not examine different time intervals between consecutive attempts) but leave it as a topic for future work. The retry threshold was chosen heuristically equal to 8, as our initial experiments suggested that after this number of retries the channel could not be successfully established within a time acceptable to the client (i.e., the total establishment time of the channel exceeds 1 sec). The metrics used in this experiment are:

- the total establishment time, and
- the load index of the established channel.

The first metric illustrates the impact of the procedure on the client, while the second illustrates the impact on the efficiency of the network. The total establishment time is the interval of time between the client's request and the network's response. The network responds to the client when

- the channel is successfully established,
- all establishment attempts have failed and the retry threshold has been exceeded, or
- the routing algorithm at any node[15] cannot determine a route that meets the real-time requirements of the channel.

In the first case the response in an *accept*, while in the other two cases it is a *denial*. The load index of the successful channel measures the resources consumed by this channel and enables us to compare the

[14] This establishment can be an initial establishment or a channel modification.

[15] In the case of retries, routing is only done at the source node.

Workload Level	Ave. Unavail. Links	Max. Unavail. Links	IR Ave. Time *(ms)*	IR Max. Time *(ms)*	Retry Ave. Time *(ms)*	Retry Max. Time *(ms)*
low	0	0	210.6	216.3	209.2	215.2
low	1	1	256.4	269.3	323.3	387.2
medium	1.8	2	311.4	330.3	441.7	481.2
high	3.6	4	364.5	396.4	534.3	573.2

Table 4.9: Experiment V Results - Establishment Times

Workload Level	IR Ave. Load Index	IR Max. Load Index	IR Hop Count	Retry Ave. Load Index	Retry Max. Load Index	Retry Hop Count
low	33.6	33.6	8	33.6	33.6	8
medium	42.8	44.0	10	33.6	33.6	8
high	54.4	56.4	14	46.6	48.4	12

Table 4.10: Experiment V Results - Load Index

efficiency of the two procedures by comparing the resource consumptions of the channels established by each procedure. In this experiment we examine the procedures at all three workload levels (*low*, *medium*, and *high*). The network topology used in this simulation experiment is as shown in Figure 4.1. The channel has the following parameters: $Xmin = 5.0$ ms, $Xave = 5.0$ ms, $I = 500$, and $D = 190$ ms, with the source at host 0 and the destination at host 24. The propagation delay of the shortest path between the source and destination is 80 ms. All of the workloads were generated so that at least one link should be unavailable along the initial route chosen by the routing algorithm. Within each workload group, 10 experiments were conducted with different random number seeds, so that a different set of requests were presented to the network but the channel classes and distribution remained the same[16].

The results of these experiments are given in Tables 4.9 and 4.10. As can be seen from Table 4.9, the *Intelligent restart* (IR) establishment method performs better across all workloads, in terms of establishment time, than multiple retries. As a baseline measurement, a set of simulation experiments were done using both establishment procedures, to determine the establishment time when there are no unavailable links. These values are provided on the first line of Table 4.9. At a *low* workload level the maximum number of unavailable links encountered during establishment across all experimental runs was 1 link, and on average the IR procedure took 20% less time than the retry procedure. This reduction in time is due to the fact that in the IR procedure the "continued" establishment attempt begins at the

[16]The load indices from these different workload sets were within 15 units of each other.

node preceding the failed link, whereas in the retry procedure the establishment message travels back to the source with the information on the failed link and then the "continued" establishment begins. The additional time due to returning to each node along that portion of the route that has been established and consequently to the source contributes to the increased time in the retry procedure. The same pattern is observed at the *medium* and *high* workload levels, where the IR procedure's average establishment time is 29.5% (*medium*) and 31.7% (*high*) less than that of the retry procedure. However, this increased performance is not free. As shown in Table 4.10, at *high* workload levels, the IR establishment procedure may result in channels that consume more resources that those established with the retry procedure. At the *low* workload levels the resources used by both of the schemes are the same, as the network is underutilized and many available links exist at each node. Also, as our topology is somewhat dense, there are several alternate links that can be chosen from the node with the unavailable link. At the *medium* workload levels the network is more utilized, and less links are available from the node with the unavailable link; hence, a longer route around this unavailable link may be needed. The routes used by the retry procedure are also shorter, as the routing algorithm does not have to consider the resources already reserved by the channel; rather, it selects the "best" route from the source to the destination. At *high* workload levels the network is near capacity; so, the routes (if indeed routes are available) are circuitous to avoid unavailable links. This is shown in Table 4.10, where two additional links are needed to avoid unavailable links. It should be noted that in all of these experiments we utilized scenarios in which the channels could be established. We suspect there may be cases at very *high* workload levels (in our context, loads with a load index value greater than 1450) where the IR procedure would be unable to establish a channel, as the final route is so long that the delay bound cannot be guaranteed, while the retry procedure could establish the channel. The retry procedure could establish the channel as the new route that is chosen does not have to consider previous links on which resources have been reserved. These initial results suggest that at *low* and *medium* workload levels the IR procedure is more suitable, but at a *high* workload level it may be advantageous to use the retry procedure. This experiment was an initial one; further simulation studies need to be done to determine the relative performances when a large number of establishments take place simultaneously. As multiple channels are being observed in these simulation experiments, the results will depend heavily on the class of each channel, the distribution of these clients' initial requests, and the times at which the initial requests occur. Essentially, the results depend heavily on the DCM **policy**. These further simulations experiments and analyses are topics for future work.

Chapter 5

Implementation

In the preceding two chapters, Chapters 3 and 4, we described the DCM scheme in detail and presented simulation experiments that verified and analyzed its functionality. In this chapter, we describe our implementation of this dynamic resource management scheme in a local area environment. A functional management scheme should enable the monitoring of relevant network variables and the control of relevant network parameters. In the context of our scheme this means monitoring the state of the guaranteed performance connections in the network and controlling these connections. This control entails the establishment of new connections and the modification of the traffic characteristics, the performance parameters, and the routes of current connections. This implementation provides the fundamental building blocks upon which the higher functional aspects of network management can be built. The OSI model of network management defines these higher functional aspects to be security management, configuration management, fault management, performance management, and accounting management [47]. These functionalities will reside on top of the DCM scheme and are in effect the DCM *policies* which will instruct the DCM scheme to collect the relevant network data and will determine the new traffic characteristics, performance parameters, or routes if a *control* action needs to be taken. This chapter is divided into two main sections, monitoring and control. The *Simple Management Network Protocol-version 1* (SNMPv1)[43] is used as the management protocol to monitor and control real-time connections. The data delivery protocols that we use to provide guaranteed-performance service are the Tenet real-time protocols; namely, a transport-level protocol, the *Real-Time Message Transport Protocol* (RMTP), and a network-level protocol, the *Real-Time Internet Protocol* (RTIP). SNMPv1 uses Management Information Bases (MIBs), which provide a hierarchical database organization model of managed information in which management data is recorded in variables. The variables in the MIBs can be read to facilitate the monitoring functionality or written to initiate a control action. In this implementation, MIBs have have been defined for the data delivery protocols (RMTP and RTIP) and for the resource management scheme (DCM). These MIBs are used by SNMPv1 to monitor and control the real-time channels.

We begin this chapter by providing some relevant background information on our environment (i.e., an overview of the Tenet Real-Time Protocol Suite and of SNMPv1) in Section 5.1. We then present a detailed discussion of the monitoring capabilities of the implementation in Section 5.2. This presentation includes the definition of the RMTP and RTIP MIBs. Section 5.3 presents the control aspects of the implementation, followed by an in-depth discussion of the SNMPv1 operations and the resource management MIB (i.e., the DCM MIB). Section 5.4 describes the initial experiments that were conducted on this prototype implementation. We conclude this chapter with a brief summary in Section 5.5.

5.1 Environment

The Tenet Real-Time Protocol Suite

The Tenet Real-Time Protocol Suite is a set of communication protocols designed to provide guaranteed-performance services in packet-switched internetworks. The components of this suite and their inter relationships are depicted in Figure 5.1. In the suite there is a clear separation between the data delivery and the control functionalities. In Figure 5.1 the portion of the stack on the left reflects the traditional layering of network protocols and is concerned with the function of data delivery. The control functions reside in the protocol on the right of the figure (the Real-Time Channel Administration Protocol (*RCAP*)), with the arrows going to and from RCAP representing the flow of channel administration control.

The data delivery functionality is achieved by the protocol stack on the left of the figure, and consists of a network-layer protocol, the Real-Time Internet Protocol (*RTIP*), and two transport-layer protocols, the Real-Time Message Transport Protocol (*RMTP*), and the Continuous Media Transport Protocol (*CMTP*).

The Real-Time Internet Protocol (RTIP) [58] provides for connection-oriented, guaranteed-performance, unreliable delivery of packets by scheduling these data packets according to the resource reservations made by the control protocol RCAP. The services provided by RTIP are used by the two transport-layer protocols. The Real-Time Message Transport Protocol (RMTP) [58] provides connection-oriented, guaranteed-performance, unreliable delivery of messages. This is a lightweight transport layer, as it only performs the functions of flow control (which is accomplished by the use of rate control) and the fragmentation and reassembly of messages. The Continuous Media Transport Protocol (CMTP) supports the transport of periodic network traffic with performance guarantees. Using the periodic nature of continuous-media traffic, CMTP [53] can gain more effective use of the network resources. This knowl-

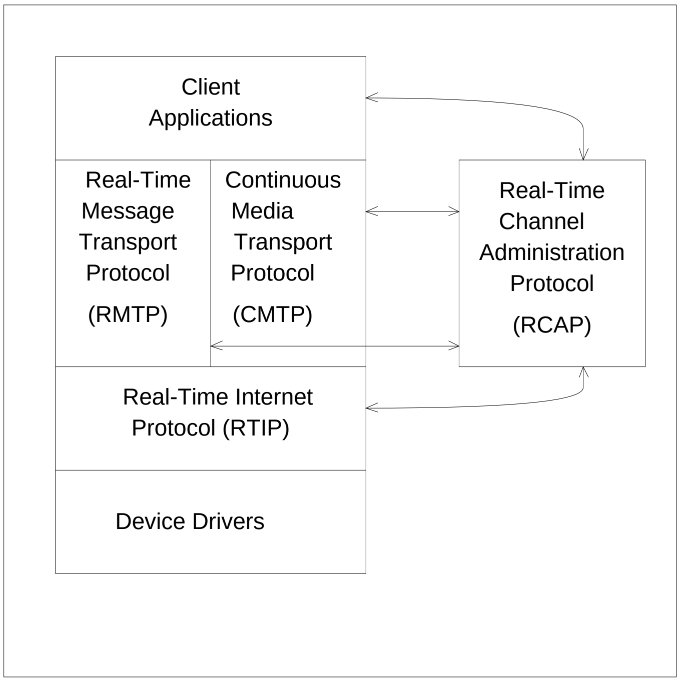

Figure 5.1: Tenet Real-Time Protocol Suite

edge is also used to implement implicit send and receive messages[1].

The control functionality in the suite is handled by the protocol on the right in Figure 5.1, the Real-Time Channel Administration Protocol (RCAP) [8]. RCAP handles channel management in response to requests from application programs. This protocol is responsible for the functions of channel establishment, teardown and status reporting. RCAP communicates with the network-layer protocol entities at each node along the channel's path, as well as with the transport-layer protocol entities at the channel's endpoints. These control paths are indicated by arrows in Figure 5.1.

The Simple Network Management Protocol - version 1 (SNMPv1)

While the Tenet Real-Time Protocol Suite does incorporate a management protocol, RCAP, our desire to examine the suitability of an "open" management system warranted the use of a standardized management framework with its associated management protocol. RCAP is a more lightweight and efficient protocol than those protocols comprising the standardized management frameworks, as it was expressly designed for speed and efficiency in real-time channel establishment. However, the goal of this implementation is not to expressly design a management framework for speed and efficiency (although all attempts will be made to obtain these qualities), but rather to investigate the suitability of an "open"

[1]Data is passed to and from the client application using shared memory; explicit send and receive system calls are not needed, as both the client application and the operating system know when data needs to be sent or to be received.

system to support real-time services.

Towards this end we examined the two most suitable management frameworks, the *Simple Network Management Protocol* [43] and the *OSI Systems Management Standards* [42]. The choice of SNMP was motivated by the market acceptance and widespread dissemination of this framework as well as by the simple, yet adequate, functionality provided by SNMP. A third consideration was the availability of source code for SNMPv1, under Ultrix4.2a, for the DECstations that comprise our local testbed as well as the SEQUOIA 2000 [48] testbed.

Simple Network Management Protocol (SNMP) is actually a collective term used to refer to a set of specifications for network management that include the protocol itself, a database definition, and some associated concepts. SNMP was designed to be an easily implementable, basic network management tool that would be used to meet the short term network management needs, with the long term solution being the *OSI Systems Management Standards*. The SNMP set of standards provides the definition of management information and a protocol for the exchange of that information. The model assumes a manager/agent paradigm in which the manager is a software module in a management system responsible for managing all or part of the network configuration and the agent is a software module in a managed device responsible for maintaining local management information and performing various actions[2]on behalf of the manager. Information exchange can be initiated by the manager (by polling) or by the agent (in response to a trap).

Within the SNMP framework, management information is represented using the *Abstract Syntax Notation One (ASN.1)*. A Management Information Base (MIB) consists of a collection of objects, each of which holds values that represent managed resources or variables. The allowable ASN.1 types and MIB structures are specified in a standard known as the Structure of Management Information (SMI). The framework also includes the specification of a set of objects that are standardized for use in all implementations, referred to as MIB-II; however, other MIBs may be defined for specific applications.

In SNMP the only operations supported are the alteration and the inspection of variables. Three operations may be performed on scalar objects:

- a **get** operation, which is issued by a manager to retrieve a scalar-object value from an agent;
- a **set** operation, which is issued by a manager that updates a scalar-object in an agent;
- a **trap** operation, which causes an agent to send an unsolicited scalar-object value to a manager.

These operations result in the exchange of management information. The protocol used to communicate between the managers and the agents is a basic mechanism for the exchange of management information. The basic unit of exchange is a message, which contains the inner Protocol Data Unit (PDU) and some

[2]These actions may be the delivery of local information or the performance of specified functions at the managed device.

control information. The control information includes a *community name*, which allows the agent to regulate access to its MIB. Five types of PDU may be carried in a SNMP message. They are :

1. the **GetRequest** PDU (issued by a manager), which includes a list of one or more object names for which values are requested;

2. the **GetNextRequest** PDU (issued by a manager), which includes a list of one or more object names; in this case, for each object named, a value is to be returned for the object that is lexicographically next in the MIB;

3. the **SetRequest** PDU (issued by a manager), which includes a list of one or more objects to be altered and their new values;

4. the **GetResponse** PDU (issued by an agent), which provides the values of the objects requested in the **GetRequest** and **GetNextRequest** PDUs, or the new values of the objects that were requested to be "set" in the **SetRequest** PDU;

5. the **Trap** PDU (issued by an agent), which provides information to a specified manager concerning an event.

These messages are sent using the User Datagram Protocol (UDP) with SNMP agents and managers listening at "well-known" ports.

5.2 Monitoring

In this section we will address the monitoring functionality that has been implemented for the data delivery portion of the real-time protocol suite. The goal of this monitoring functionality was to provide the ability to monitor the guaranteed-performance connections that will be established and modified in integrated-services packet-switched networks. This monitoring capability will enable us to support the control functionality that comprises the high-level management functionalities described in the OSI model. As described in Section 5.1, message data delivery[3] is accomplished by two protocols: RTIP, which is a network-level protocol, and RMTP, which is the transport-level protocol. As our real-time connections are virtual circuits, the network-level protocol, RTIP, requires state information to be kept in each of the nodes or switches that a connection traverses, whereas the transport-level protocol, RMTP, requires state information only in the source and destination hosts (i.e., the end points of the connections).

[3]In this work we are only concerned with the monitoring and control of RMTP streams as opposed to CMTP streams.

The MIBs for RMTP and RTIP are provided below. Figure 5.2 provides an overview of the current structure of the MIB. We have elected to place our real-time MIBs in the private/enterprise subtree. It should be noted that we have not acquired permission for the enterprise number in the figure, so this number may indeed change in the future to accommodate legal users. The TenetGroup subtree (1.3.6.1.4.1.18) is the root of our MIB, and has one subtree called Scheme1 (it is hoped that Scheme2 will in the future become another subtree to TenetGroup). Under the Scheme1 subtree (1.3.6.1.4.1.18.1) are the three subtrees that support the monitoring and control of real-time connections.

Figure 5.3 shows the RTIP group, which consists of 3 scalars and a table that are described below. The 4 objects in this group are:

1. **RtipConnTable:** A table of RTIP channel-specific runtime information.
2. **RtipVersionID:** The current RTIP version number.
3. **RtipMaxConn:** The maximum number of permissible RTIP channels.
4. **RtipNumConn:** The number of current RTIP channels at this node.

The **RtipConnTable** table is comprised of instances of the object **RtipConnEntry**. This object is a **SEQUENCE** of the objects described below.

1. **RtipLcid:** The local channel identifier at this node (index).
2. **RtipConnStat:** The status of the channel (e.g., the source, destination, or intermediate node).
3. **RtipInPkts:** The number of incoming packets on the channel (i.e., packets that have been received at the node).
4. **RtipInLatePkts:** The number of incoming packets that were late at this node.
5. **RtipInHdrErr:** The number of incoming packets with header errors.
6. **RtipOosPkts:** The number of incoming packets that were out of sequence.
7. **RtipInChkErr:** The number of incoming packets with header checksum errors.
8. **RtipForPkts:** The number of packets that were forwarded at this node.
9. **RtipOutPkts:** The number of outgoing packets over this channel.

The first variable in the **RtipConnEntry** object, **RtipLcid**, provides the **INDEX** for the table, as it uniquely identifies each object instance. The Structure of Management Information (SMI), which

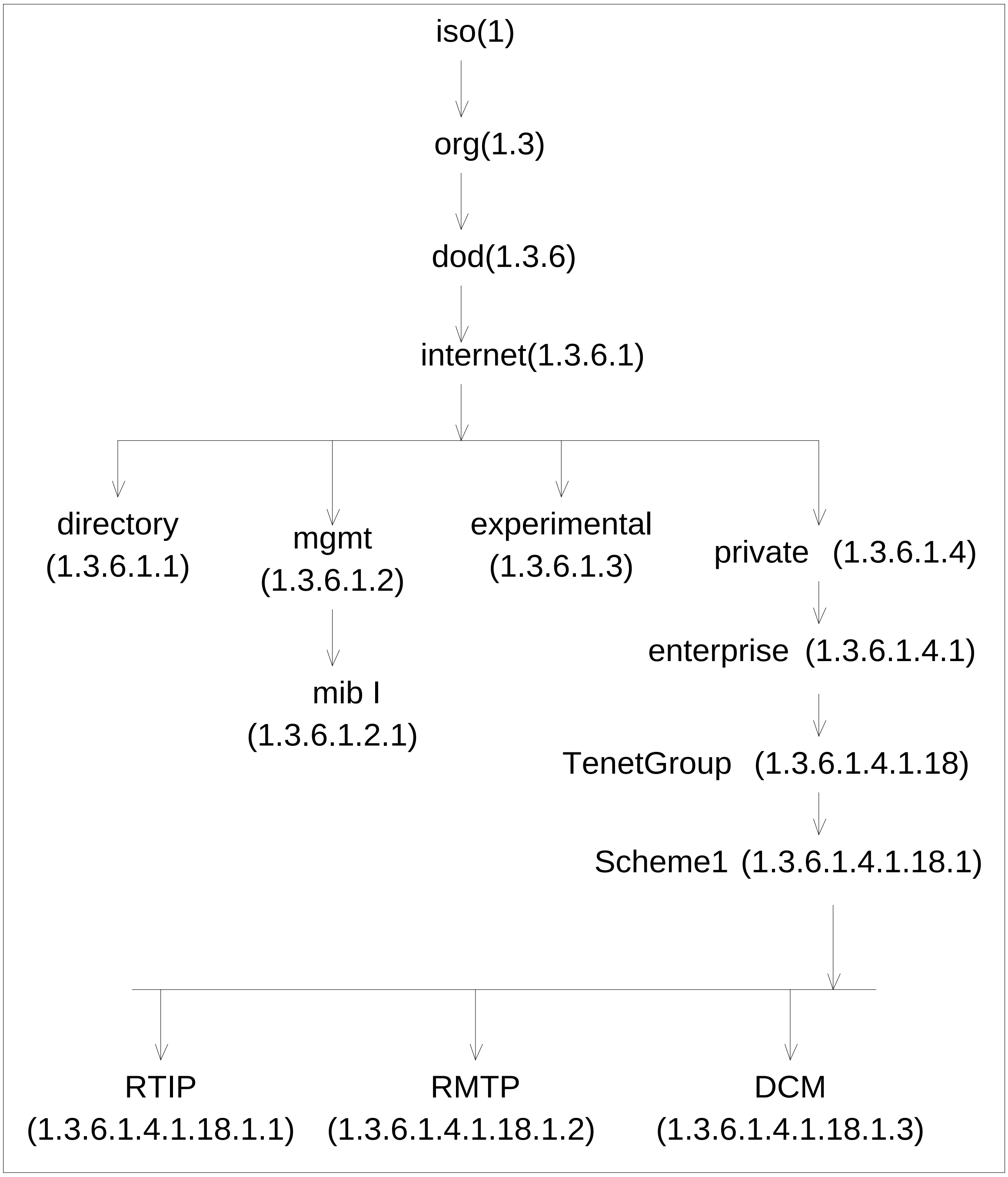

Figure 5.2: SNMP MIB-I

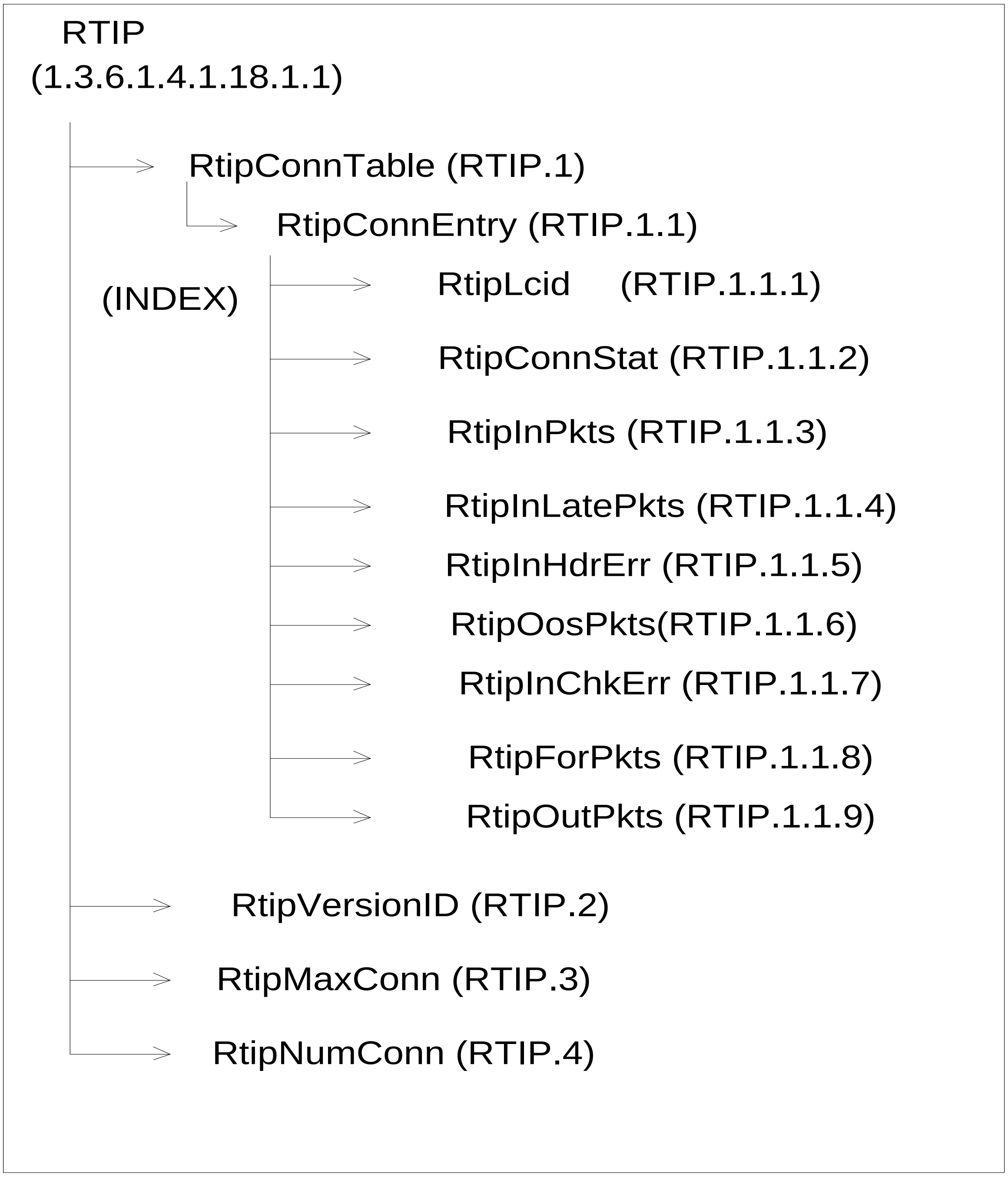

Figure 5.3: RTIP Managed Objects

specifies the structure and allowed ASN.1 types of these objects in the MIB, is provided in Appendix I. It should be noted that all of the information contained in this table has its **ACCESS** defined as Read Only (RO) and its **STATUS** as mandatory (M). The mandatory status indicates that in any node supporting this MIB all of the objects in the RTIP group must be supported.

The RMTP MIB is shown in Figure 5.4 and consists of 3 scalars and a table similar to that described for the RTIP MIB. This MIB is active only at the end points of the connection.

The 3 objects in this group are:

1. **RmtpConnTable:** A table of RMTP channel-specific runtime information.

2. **RmtpMaxConn:** The maximum number of permissible RMTP channels.

3. **RmtpNumConn:** The number of current RMTP channels at this node.

The **RmtpConnTable** is comprised of instances of the object **RmtpConnEntry**. This object is a **SEQUENCE** of the objects described below.

1. **RmtpSourceLcid:** The local channel identifier that was specified at the source node (index).

2. **RmtpLocalChannelAddr:** The IP address of the source node (index).

3. **RmtpInMess:** The number of incoming messages for this channel (i.e., the number of messages received).

4. **RmtpInReassErr:** The number of reassembly errors that occurred in processing the incoming messages.

5. **RmtpOutMess:** The number of outgoing messages from this channel.

6. **RmtpOutErr:** The number of outgoing messages that could not be sent due to fragmentation errors.

The first 2 objects provide the **INDEX** to the **RmtpConnTable** table, and can be used to uniquely access each instance of the **RmtpConnEntry**. This MIB also has its **ACCESS** as Read Only (RO) and its **STATUS** as mandatory (M). Its SMI is also be provided in Appendix I.

The SNMP **Get** or **GetNext** commands are used to access these objects. A connection can be *walked* by accessing the entry in the **DcmConnTable** table in the DCM group (this group is discussed in Section 5.3.4). The DCM group contains the objects **DcmRoute** and **DcmRouteLcids**, which provide the nodes that comprise the route and the local channel identifiers (i.e., the **RtipLcid**) at each of these node. After obtaining this *static* information, another **get** command is used to obtain the

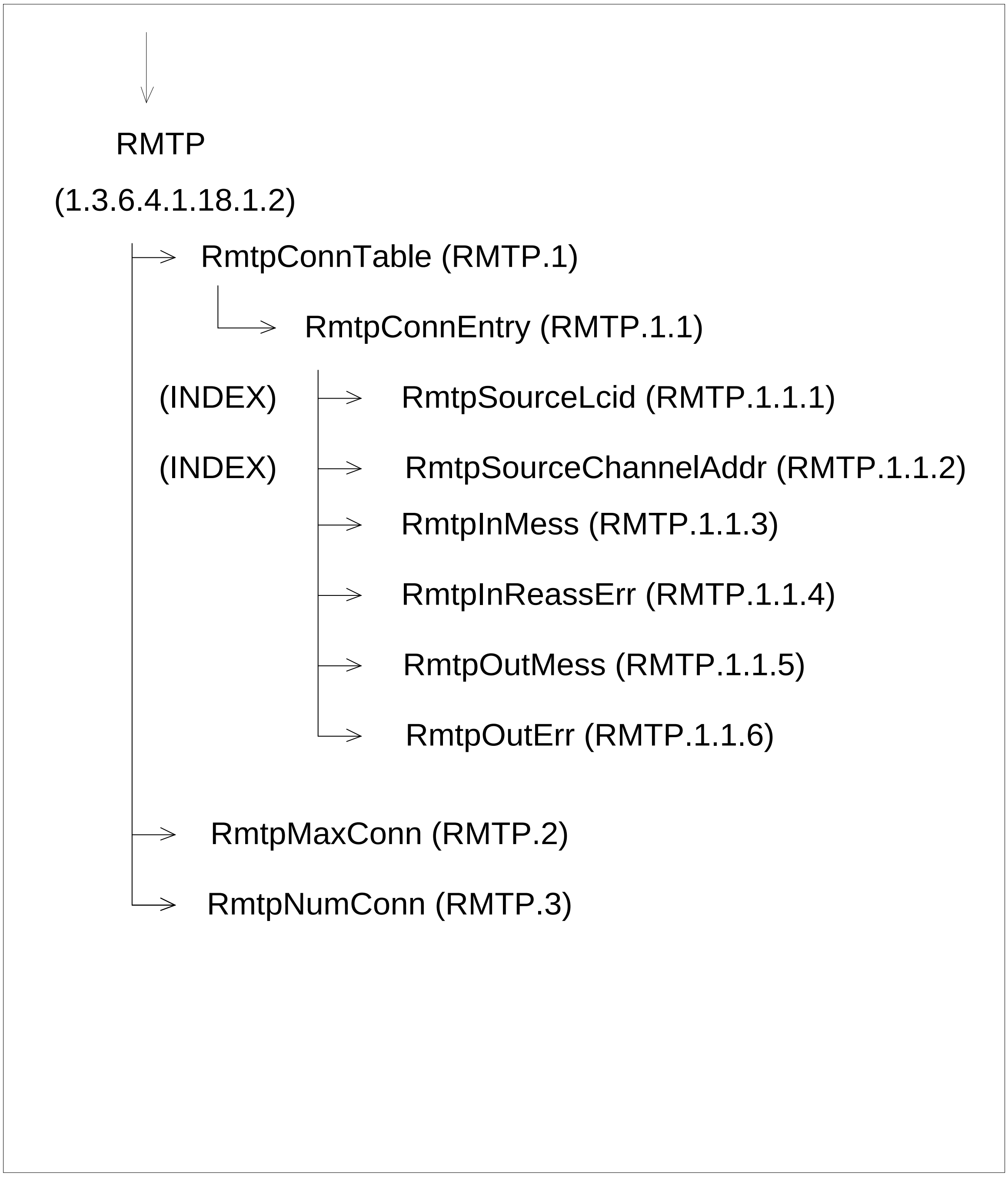

Figure 5.4: RMTP Managed Objects

runtime information from the **RtipConnTable** table, specified by the *static* information, on any node along the path.

It should be noted that, due to the manner of the DEC SNMPv1 agent implementation, modifications were needed in RMTP and RTIP to support these MIB definitions. Also, these MIBs constitute our initial attempts at monitoring real-time connections and, as such, represent a base set of monitoring variables. With the increased usage of these guaranteed-performance data delivery protocols, we expect that the format of their MIBs will change to reflect the needs of the network managers and other client applications.

5.3 Control

5.3.1 Overview

In this section we present the control capability of our DCM implementation. We begin by describing the control flow in establishing, modifying, and terminating a channel. SNMPv1 is our management or control protocol, and is used to exchange resource reservation messages among nodes along the path of the channel. The three interfaces to this "reservation" protocol, namely, the *Application Client/DCM Policy Manager interface*, *the DCM Policy Manager/SNMP Agent interface*, and *the SNMP Agent/RTIP interface*, are then defined. Finally, the resource management MIB, i.e., the Dynamic Connection Management (DCM) MIB, is presented.

Guaranteed-performance connections are established, modified, and terminated by using the **set** command of SNMPv1. The "setting" of a sequence of objects reflects the establishment or modification of a channel with the traffic characteristics and performance requirements corresponding to the values of the managed objects that are being set. The **set** command is atomic in that, if a subset of these variables cannot be set, then none of the objects in the sequence will be set. This corresponds to the situation in which a subset of the traffic characteristics and performance requirements requested by the client cannot be honored due to resource unavailability, and hence the connection cannot be established or modified. The termination of a channel is achieved by "setting" the values of this sequence of objects to "well-known" *NULL* values. The establishment or modification of a connection consists of setting the appropriate objects at each node along the path traversed by the connection. If at any node along the path these objects cannot be set (this situation may occur due to the lack of resources at this node or because the client does not have permission to access this node), then the connection is refused, and all of the previously reserved resources are released.

It should be noted that the establishment and modification functions are usually implemented with signaling protocols that are reliable, peer-to-peer protocols, whereas SNMPv1 is a master/slave protocol, with SNMPv1 agents subservient to SNMPv1 managers. While SNMPv2 [47] supports peer-to-peer manager communication, this protocol is not at a suitable stage of evolution to be considered in our development effort. To our knowledge, only a few prototype implementations of SNMPv2 on Ultrix4.2a exist, and currently they are not stable. As we mentioned in Section 5.1, the other choice of a framework was the OSI management framework, which does support peer-to-peer communication; however, the complexity of this framework, the lack of available implementations, and the desire for a quick prototype rendered this choice unacceptable. Therefore, in order to incorporate some of the functionality provided by signaling protocols while trying to maintain the simplicity inherent in SNMPv1, we have made two relevant changes to the structure of the SNMPv1 operations. These two changes are :

- an agent's response (i.e., the GetResponse PDU) can be sent to the requesting manager and/or to another manager (currently in SNMPv1 the GetResponse PDU can only be sent to the requesting manager)[4], and
- the underlying unreliable transport protocol for SNMPv1 (i.e., UDP) has been made reliable by using sequence numbers, timeouts, and retransmissions.

The complete protocol stack is shown in Figure 5.5.

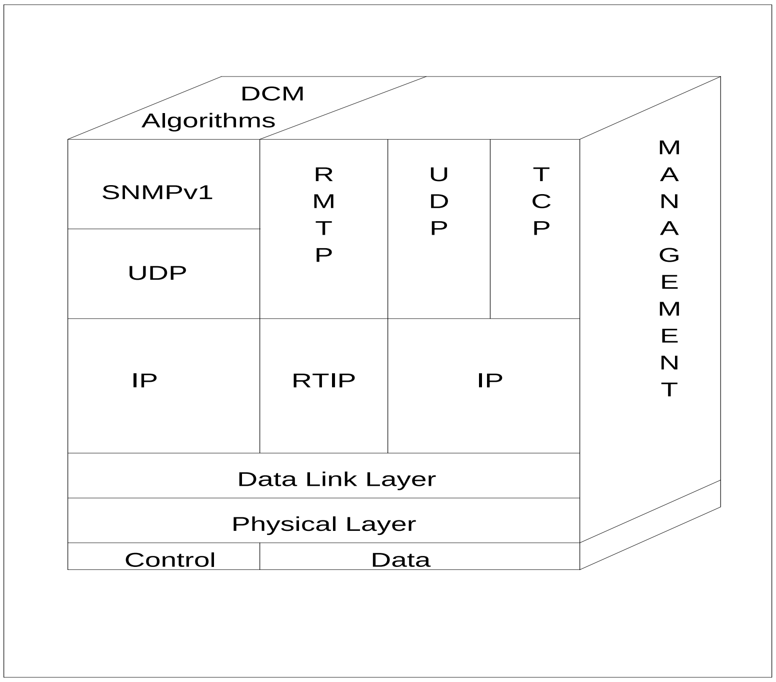

Figure 5.5: Protocol Stack

[4]This functionality is realized in SNMPv2 by using the **InformRequest PDU**.

5.3.2 Controlling a Real-Time Connection

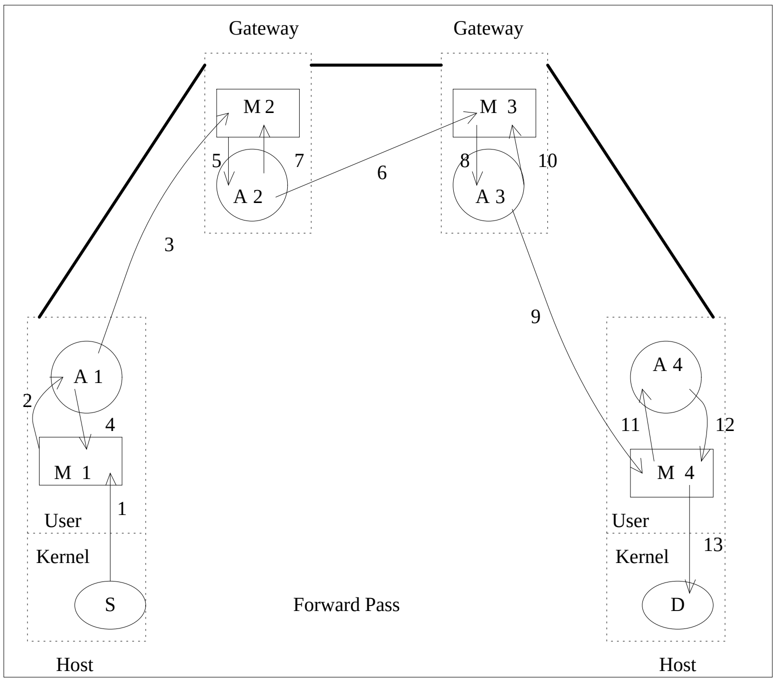

Figure 5.6: Connection Establishment/Modification - Forward Pass

Figures 5.6 and 5.7 show the control message flows (for the forward and reverse passes, respectively) during the establishment and modification of a guaranteed-performance connection. These flows take the form of either socket calls between the client's sending or receiving application and the DCM Policy Manager (M), or SNMP messages between the DCM Policy Manager and the SNMP agents (A). Connection establishment and modification are achieved in two passes (one round trip), a forward or *reservation* pass and a reverse or *relaxation* (i.e., release) pass. Both connection establishment and modification result in the same message flow[5]; however, there are some differences between them. In connection modification, the routing algorithm may need to compensate for the resources reserved for the current connection (if resource sharing is used), the admission control tests that are conducted are different, additional buffers may need to be reserved at the destination node for re-sequencing, packets may need to be resequenced at the destination, and the relaxation routine is different. None of the above differences affect the message flow, but the content of the messages is different. Connection termination is also accomplished in a round trip with resources being released on the forward pass and data structures cleared on the reverse. The message flow is identical to that of connection establishment or modification, with the message content indicating a connection termination. The content of the various messages is

[5] *Global* and *Local* modifications of connections use the same control message flow, although *local* modifications are across a portion of the route.

given in Section 5.3.4. The forward pass is depicted in Figure 5.6. The message flows are described in numbered order below:

1. This message results from a socket call (discussed in Section 5.3.3), which generates an upcall to the DCM policy manager (M1) and provides the manager with the parameter and control information for channel establishment or modification. The DCM Policy manager then examines the client's request and determines if it should be attempted by examining its *Policy Rules*. If an attempt is to be made, it uses the DCM routing algorithm to determine a route for this message.

2. This message results from an SNMP **set** command from the policy manager (M1) and takes the form of a **SetRequest** PDU that is sent to the local SNMPv1 agent (A1). The local agent verifies the message using the *community field* in the message header, and then conducts admission control tests using the parameters contained in the **SetRequest** PDU. If the admission control tests are successful, the agent reserves the appropriate resources to accommodate the connection at this node (i.e., it reserves the appropriate bandwidth and schedulability resources, and buffer space needed for this connection). A detailed description of the actions taken by the local agent is provided in Section 5.3.4.

3. If the admission tests succeeded at this node, the local agent (A1) sends a **GetResponse** PDU to the **downstream policy manager** (M2) containing the objects to be set, their new values, and some additional updated information. This updated information reflects the resources reserved in the local node and other state information. As mentioned previously, this action corresponds to a change in the usual SNMPv1 operations, as the **GetResponse** PDU is usually sent only to the local manager. The choice of the remote policy manager, upstream or downstream, is based on the result of the admission control test (acceptance or denial), the direction of the establishment message (forward or reverse pass), and the current node at which we are executing. In Figure 5.6, we assume that the admission tests are successful along the forward pass; hence, a **GetResponse** PDU is passed to the next downstream policy manager. If the response had been a denial, a **GetResponse** PDU would have been sent to the upstream policy manger. The failed connection scenario is provided in Figure 5.8, and will be explained later in this section. It should also be noted that the reliability that we have provided may result in message 3 being retransmitted until an acknowledgment is received. This acknowledgment and possible retransmission can occur simultaneously with message 4. For the sake of simplicity, in Figure 5.6, this reliability is implied and only message 3 is depicted.

4. The local agent (A1) then sends the identical **GetResponse** PDU (sent in message 3) to its local policy manager (M1).

5. The policy manager (M2) then examines the **GetResponse** PDU to see if the request is permitted. If so, it issues an SNMP **SetRequest** PDU with the same contents as the **GetResponse** PDU to its local agent.

6. The local agent (A2) does the admission control tests to determine if the connection can be admitted. If admitted, resources are reserved accordingly. Assuming that the admission tests are successful, the agent sends a **GetResponse** PDU to the next downstream policy manager (M3) indicated in the **set** request. This **GetResponse** PDU also contains updated information pertaining to the locally reserved resources.

7. The local agent (A2) sends a **GetResponse** PDU to its local policy manager (M2).

8. The policy manager (M3) then examines the **GetResponse** PDU to see if the request is permitted. If so, it passes a **SetRequest** PDU with the same contents as the **GetResponse** PDU to the local agent (A3).

9. The local agent (A3) does the admission control tests and reserves resources accordingly. Based on the results of the admission tests, this agent sends a message to either the downstream or upstream policy manager and its local policy manager. In this case (a successful admission control test), the agent sends a **GetResponse** PDU to the next downstream Policy Manager (M4) with the updated information.

10. The local agent (A3) sends a **GetResponse** PDU to its local policy manager (M3).

11. The policy manager (M4) then examines the **GetResponse** PDU to see if the request is permitted. If so, it sends a **SetRequest** PDU with the same contents as the **GetResponse** PDU to the local agent (A4).

12. The local agent (A4) does the final computation to determine if the channel is accepted, and sends a response message to its DCM policy manager (M4). This is the one exception to the previous operations in that, at the destination node, the **GetResponse** PDUs from the agent is only sent to the local policy manager.

13. The DCM policy manager (M4) informs the receiver that a request is pending. The receiver can then decide if it wishes to accept the connection; based on its decision, the DCM policy manager then begins the message flow along the *reverse* pass.

Note that the DCM policy manager does not have to be at the same local node as the agent, as the manager can send SNMP **SetRequest** PDU to the agents from remote locations. This is the case in network-initiated control.

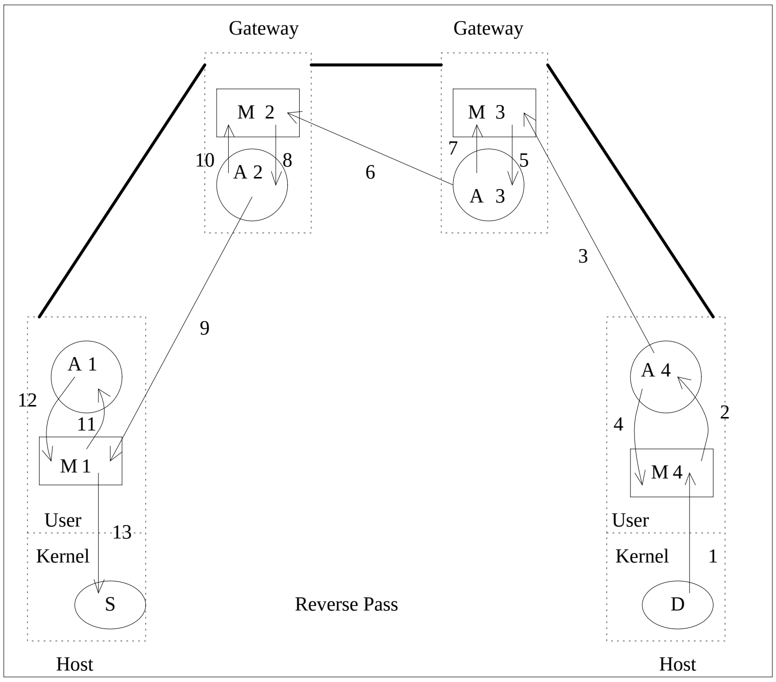

Figure 5.7: Connection Establishment - Reverse Pass

The reverse pass of the establishment/modification message is depicted in Figure 5.7. The path followed by the establishment message on this reverse pass is always the same regardless of the state of the SNMP message (i.e., regardless if the connection was accepted or denied); however, the contents of the message differ depending on this state. The message flow is described below.

1. The receiver returns its decision to its policy manager (M4). This decision can be either an acceptance or a denial of the connection. In this example, the decision is acceptance.

2. The manager (M4) sends a **SetRequest** PDU with the state variables indicating the acceptance of the connection to the local SNMPv1 agent (A4) and the relaxed local delay values. This local agent then calculates the relaxed values for the delay bounds of the links, and then fills in the appropriate SNMPv1 data structures that are used in the RTIP, RMTP and DCM MIBs.

3. The local agent (A4) sends a **GetResponse** PDU with the state variable indicating acceptance and providing the relaxed delay bound values to the upstream policy manager (M3).

4. The local agent (A4) also returns the same **GetRequest** PDU to the local policy manager (M4).

5. The upstream manager (M3) examines the **GetResponse** PDU and then formulates a **SetRequest** PDU that is sent to the local agent (A3).

6. This local agent (A3) receives the message from the local policy manager (M3) and, if necessary, adjusts the reserved resources to reflect the new relaxed delay bound values. The agent then sends a **GetResponse** PDU to the upstream policy manager[6] (M2).

7. The agent (A3) also returns the same **GetRequest** PDU to the local policy manager (M3).

8. The policy manager (M2) examines the **GetResponse** PDU, and then formulates a **SetRequest** PDU that is sent to the local agent (A2).

9. This local agent (A2) receives the PDU from the local policy manager (M2) and, if necessary, adjusts the reserved resources. The agent then sends a **GetResponse** PDU to its upstream policy manager (M1).

10. The agent (A2) also returns the same **GetRequest** PDU to the local policy manager (M2).

11. The policy manager (M1) examines the **GetRequest** PDU and then sends a **SetRequest** PDU to the local agent (A1).

12. This local agent (A1) receives the message from the local policy manager (M1) and, if necessary, adjusts the channel's reserved resources. The agent then returns a **GetResponse** PDU to its local policy manager (M1).

13. The policy manager (M1) then does a downcall to unblock the sending client, and provides the response to the client.

Figure 5.8 depicts the scenario in which a connection establishment or modification request is refused at an intermediate node due to unavailable resources or the absence of the appropriate permissions. The forward pass messages are depicted by the un-numbered dotted arrows and proceed (in the manner described previously) to the second gateway, while the numbered solid arrows show the reverse pass failure messages. The message flow resulting from the failure proceeds from the node at which the failure takes place:

1. The local agent (A3) does the admissions control tests and the result is failure. This local agent then sends a **GetResponse** PDU to the upstream policy manager (M2) indicated in the previous **SetRequest** PDU. This PDU has the status bit set to reflect denial and the reason code field updated accordingly.

2. The local agent (A3) sends a **GetResponse** PDU to its local policy manager (M3).

[6] As before, reliability is implicit in this message flow, and only the initial message flow is shown.

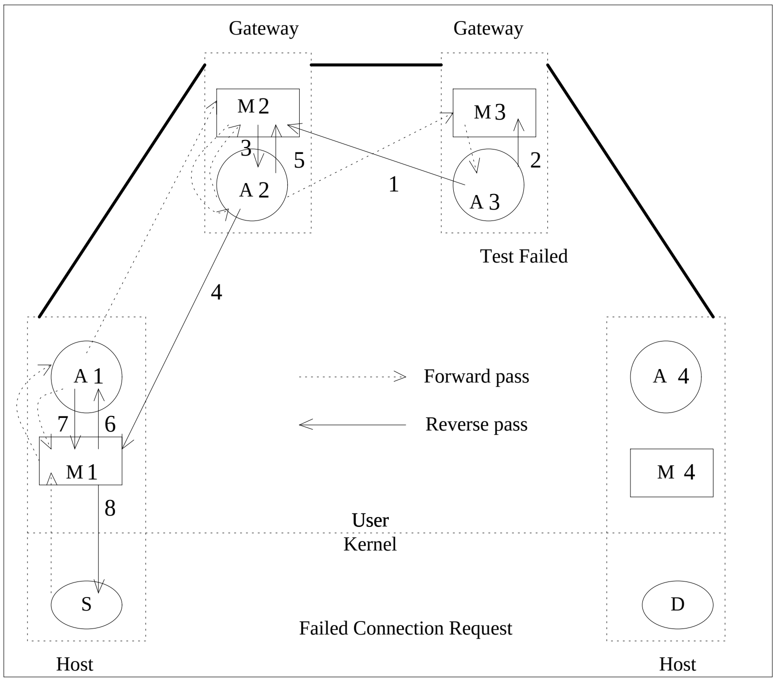

Figure 5.8: Failed Connection Establishment

3. The upstream manager (M2) sends a **SetRequest** PDU to its local agent (A2) with the status bit reflecting a denied connection.

4. The local agent (A2) releases the resources reserved at the node and sends a **GetResponse** PDU to the upstream policy manager (M1) indicated in its variables.

5. The local agent (A2) then sends a **GetResponse** PDU to its local policy manager (M2).

6. The upstream manager (M1) receives the **GetResponse** PDU and sends a **SetRequest** PDU to its local agent (A1), with the status bit reflecting the denial of the connection.

7. The local agent (A1) releases the resources reserved at the node and sends a **GetResponse** PDU back to its local policy manager (M1).

8. The policy manager (M1) then unblocks the sending client's process and returns to the client the reason for the denial.

The failure scenario at the destination node (i.e., failure due to unavailable resources) is exactly the same as that at the intermediate nodes. If the failure is due to the rejection at the receiver, the information flow is the same as depicted in Figure 5.7. A failure at the sending node results only in the return message (i.e., **GetResponse** PDU) and a downcall to unblock the sending process. The connection termination procedure uses a message flow identical to that of connection establishment or modification. The functional difference is that on the forward pass the resources are released, and on the reverse pass the data structures are freed. The contents of the messages or PDUs used on these passes are discussed in Section 5.3.4.

The above scenarios depicted the client-initiated establishment and modification. In the case of the network-initiated establishment or modification, the scenario is nearly the same, with the exception that messages 1 and 13 in Figure 5.6, which are messages from the sending process and to the destination process, do not exist, and the initiating policy manager need not be local. The policy manager can be remote once it is verified by the local SNMPv1 agent as an authorized policy manager. In that case messages 2 and 4 in Figure 5.6 are from and to the remote policy manager. The remainder of the call sequence is the same as before. On the reverse pass, Figure 5.7 messages 1 and 13, which are messages from the destination process and to the sending process, do not exist, and messages 9 and 12 are to the remote policy manager.

5.3.3 Interfaces

In this section we discuss three interfaces. The first interface exists between the application client (i.e., the entity requesting a connection or modifying a connection in a client-initiated modification) and the DCM manager, the second interface exists between the DCM manager and the SNMPv1 agent, while the third exists between the SNMPv1 agent and RTIP. The first interface allows the client-initiated establishment and modification of a real-time channel, and takes the form of socket calls from the application client. The second interface is the conventional SNMPv1 interface between a manager and an agent (i.e., the **get** and **set** operations). The third interface is between the SNMPv1 agent and the data delivery protocol, and uses a socket call to initialize the RTIP data structures and to initiate the RMTP/RTIP connection.

The Client/SNMP Manager Interface

Sender's Socket Calls
1. *sockData = socket(AF_INET, SOCK_DGRAM, IPPROTO_RMTP)*
2. *bind(sockData, sender_address, addrlen)*
3. *connect(sockData, DCMParameters, length)*
4. *setsockopt(sockData, IPPROTO_RTIP, RTIP_MOD, DCMParameters, length)*
5. *select(maxfiled, NULL(R), sockData(W), sockData(E), NULL)*
6. *getsockopt(sockData, IPPROTO_RTIP, RTIP_RESP, &DCMResponse, &length)*
7. *close(sockData)*

Table 5.1: Sender's Interface

This interface consists of a sequence of socket calls that are issued by the sender or receiver at the end points of the connection. The first sequence of calls contains those issued by the sender to the DCM Policy Manager, and is depicted in Table 5.1. They are as follows (this is the sequence of calls used if the client establishes a connection and then modifies the connection):

1. *sockData = socket(AF_INET, SOCK_DGRAM, IPPROTO_RMTP).* This call creates a real-time socket for sending real-time data.

2. *bind(sockData, sender_address, addrlen).* This call binds the address provided by the client to the socket.

3. *connect(sockData, DCMParameters, length)* . This is a **blocking** call that establishes the real-time connection specified in the *DCMParameters* data structure. If the establishment is successful, this call returns a value of *-1*; if the call is unsuccessful, the call returns a value greater than zero that indicates the reason for the failure. This call causes an upcall to be generated, which sends a message to the DCM policy manager.

4. *setsockopt(sockData, IPPROTO_RTIP, RTIP_MOD, DCMParameters, length)* . This call issues a request to modify the parameters of a real-time connection, and provides those new parameters in the *DCMParameters* data structure.

5. *select(maxfiled, NULL(R), sockData(W), sockData(E), NULL).* This call allows the sender to continue sending with the original parameter values until notification has been provided that a response has arrived. This notification occurs as an **exception** on the *sockData* socket. The *select* call allows the application to write data to the socket while still permitting the DCM policy manager to inform the client when the response to the modification occurs. The policy manager's response is obtained by doing the next call.

6. *getsockopt(sockData, IPPROTO_RTIP, RTIP_RESP, &DCMResponse, &length).* This call returns the response to the modification request. If the response is success, then the application can begin sending at the new rate or expecting that the data sent from this moment reaches the destination with the new parameters. If the response is fail, then a reason code is provided in the *DCMResponse* structure.

7. *close(sockData)* . This call tears down the connection and closes the socket.

If the client does not require modifications on this connection, then only calls (1), (2), (3) and (7) are needed.

The receiver also issues a sequence of calls to allow it to receive incoming real-time connections, as shown in Table 5.2. This sequence is:

Receiver's Socket Calls
1. *sockRec = socket(AF_INET,SOCK_DGRAM,IPPROTO_RMTP)*
2. *bind(sockRec,destination_address,addrlen)*
3. *listen(sockRec,backlog)*
4. *select(maxfiled,sockRec,Other,Other,NULL)*
5. *getsockopt(sockRec,IPPROTO_RTIP,RTIP_RECREQ,&DCMParameters,&length)*
6. *newsockRec = accept(sockRec,sender_address,addrlen)*
7. *setsockopt(sockRec, IPPROTO_RTIP, RTIP_FAILRES, DCMResponse, length)*

Table 5.2: Receiver's Interface

1. *sockRec = socket(AF_INET, SOCK_DGRAM, IPPROTO_RMTP)*. This call creates a socket for receiving real-time data.

2. *bind(sockData, destination_address, addrlen)*. This call binds the destination address provided by the client to this socket.

3. *listen(sockRec, backlog)*. This call informs the DCM Manager that this port is available for real-time connections. The backlog indicates the maximum number of permissible pending conditions.

4. *select(maxfiled, sockRec, Other , Other, NULL)*. This call allows the receiver to accept data on other sockets (or do other work) while it awaits a request for a real-time connection.

5. *getsockopt(sockRec, IPPROTO_RTIP, RTIP_RECEIVE_REQUEST, &DCMParameters, &length)*. This call gets the characteristics and parameters of the connection request. The destination can then decide if it wishes to accept this call. The next 2 calls are for the cases of a destination's **acceptance** or **denial**, respectively.

6. *newsockRec = accept(sockData, destination, length)* . The request is accepted by the destination process and the reverse pass of the establishment or modification message can begin. This call blocks until all of the RTIP and RMTP data structures are established at the destination node. Data can now be received on the *newsockRec* socket.

7. *setsockopt(sockRec, IPPROTO_RTIP, RTIP_FAILED_RESPONSE, DCMResponse, length)* . This call is used when the response is denied; the reason for this denial is returned to the sender in the *DCMResponse* structure.

An additional socket call,

setsockopt(sockData, IPPROTO_RTIP, RTIP_UNREGISTERPORT, port, length), is used to un-register a port, thus preventing any connections from being established at this port. Attempts to connect to this port are returned with the appropriate error message included for the sender.

In these socket calls the traffic and performance parameters are passed to the DCM policy manager using the *DCMParameters* structure, and the manager's response returned using the *DCMResponse* structure. These two structures are provided below.

```
struct dcm_param {
struct TrafficSpec  Init_DcmTrafSpec;  /* channel's initial traffic spec */
struct PerfSpec     Init_DcmPerfSpec;  /* channel's initial perf. spec */
struct in_addr      DcmDestIpAddress;  /* Destination IP address */
int                 DcmDestPort;       /* Destination port number */
struct TrafficSpec  New_DcmTrafSpec;   /* channel's modified traffic spec */
struct PerfSpec     New_DcmPerfSpec;   /* channel's modified perf. spec */
char               *Dcm_DestMessage;   /* Information passed to the receiver */
} DCMParameters;
```

```
struct dcm_resp{
int     Dcm_response;           /* Acceptance or reason for failure */
char   *Dcm_additional_info;    /* Add. information from receiver   */
} DCMResponse;
```

In the *DCMParameters* structure the channel's initial traffic specification and initial performance specification are provided by the client in the *Init_DcmTrafSpec* and *Init_DcmPerfSpec* data structures, respectively. These structures contain the traffic and performance parameters of the Tenet performance contract as described in Section 3.1.1. *DestIpAddress* contains the Internet address of the destination node, with the *DestPort* integer containing the port number at this node. In the event that this request is a modification request (i.e., sender's socket call number 4 and the corresponding receiver's socket call number 5), the new traffic characteristics and performance parameters are provided in the *New_DcmTrafSpec* and *New_DcmPerfSpec* data structures, respectively. The final field in this DCMParameters structure is *Dcm_DestMessage*, which is a pointer to a character string that can be used to send information to the receiver. This information may be used by the receiver to determine accessibility to the receiver's service or to provide other higher level quality-of-service information.

In the *DCMResponse* data structure there are two fields. The first field, *Dcm_response*, provides an integer that indicates if the request has been accepted or a numbered reason for its denial. In the event that the denial was determined by the client's receiver application (who is the high-level service provider; for example, a video-on-demand server) the reason for this denial may be sent in the string whose pointer is *Dcm_additional_info*[7].

DCM Policy Manager/SNMPv1 Agent Interface

The interface used between the DCM policy manager and the SNMPv1 agent is the usual SNMPv1 interface comprised of the **set** and **get** operations. The use of this interface has already been demonstrated in Section 5.3.2 as the **set** requests generate **SetRequest** PDUs that are sent from the local policy managers to the local agents. The use of the **get** commands in monitoring the connections was also demonstrated in Section 5.2. This interface is exclusively used in network-initiated modifications, especially when transparent modifications are required. An example of this is load balancing or resource reclamation, in which network monitoring indicates that route modifications are required to more efficiently utilize the network. These route modifications, which can be local or global, are transparent to the client, and result from the **SetRequest** PDU sent by the controlling DCM policy manager to the local agent on the node at which the reroute begins. The message flow and contents are the same as those used in the establishment and modification of a channel with the values of the objects set to indicate local or global rerouting with the routes provided, or to indicate that the routes should be determined by that policy manager at the rerouting node.

The SNMPv1 Agent/RTIP Interface

The interface used here is the conventional interface used by the Tenet protocols to communicate between RCAP (the control protocol) and RTIP (the data delivery protocol). In this interface, the SNMPv1 agent uses a pre-defined **control** socket to update the relevant variables in the RTIP kernel tables. This interface, which is detailed in [55], is:

- *setsockopt(sockControl, IPPROTO_RTIP, RTIP_SPEC, RtipSpec, length).* This "non-blocking" call writes the RTIP specifications into the kernel level RTIP table, thereby establishing the data path of the connection through this node. The RTIP table is used to determine the level of service, the outgoing link, and other control actions to be taken on each data packet.

5.3.4 The Resource Management MIB

In this subsection we will provide the management information base for the DCM group. This MIB contains aggregate and specific data on the state of resource management in the node. This data includes the state of the real-time connections (i.e., the traffic and performance specifications, the status, and

[7]This string would allow user data to be sent back to the client's sending application.

other relevant information) traversing the node, aggregate information on the number of requests at this node, and the state of the node resources (i.e., the reserved amounts of the bandwidth, delay, and buffer space resources).

Figure 5.9 provides the managed objects of the DCM group. This group contains 10 scalars and 2 tables as described below. The first object in this group is the **DcmConnTable** table, which contains the state of each real-time channel traversing the node. The next 7 scalar objects contain the aggregate information on requests at this node. As this is a prototype MIB for the DCM scheme, only a small subset of the possible request information was recorded, namely the number of requests, the number of successful requests, and the number of requests that failed because of the unavailability of a route. Route unavailability occurs when the routing algorithm determines that no route is available that satisfies the traffic and performance characteristics needed by the requesting client, given the algorithm's knowledge of the current network load. The next scalar, the *DcmEnableAuthTrap* object, is a **trap** object that allows the local agent to generate an un-initiated report to a specified policy manager indicating that an unauthorized manager had attempted to do a control action at this node. The next object is a table, **DcmRcspTable**, whose entries provide the resources reserved at each priority or delay level in the RCSP server at this node. The final two objects are scalars that provide the total buffer capacity and the amount of this buffer capacity that is currently reserved. The objects in the DCM group are:

1. **DcmConnTable:** A table of channel-specific management information.
2. **DcmTotalEstReq:** The number of establishment requests received.
3. **DcmSuccEstConn:** The number of establishment requests that were accepted.
4. **DcmTotalModReq:** The number of modification requests received.
5. **DcmSuccModConn:** The number of modification requests that were accepted.
6. **DcmFailModRoute:** The number of modification requests that failed due to route unavailability.
7. **DcmFailEstRoute:** The number of establishment requests that failed due to route unavailability.
8. **DcmStatReq:** The number of connection status requests received.
9. **DcmEnableAuthTrap:** This object enables or disables a trap that will indicate an unauthorized control attempt.
10. **DcmRcspTable:** A table of current RCSP reservation information.
11. **DcmTotalBufCapacity:** The total buffer capacity at this node.

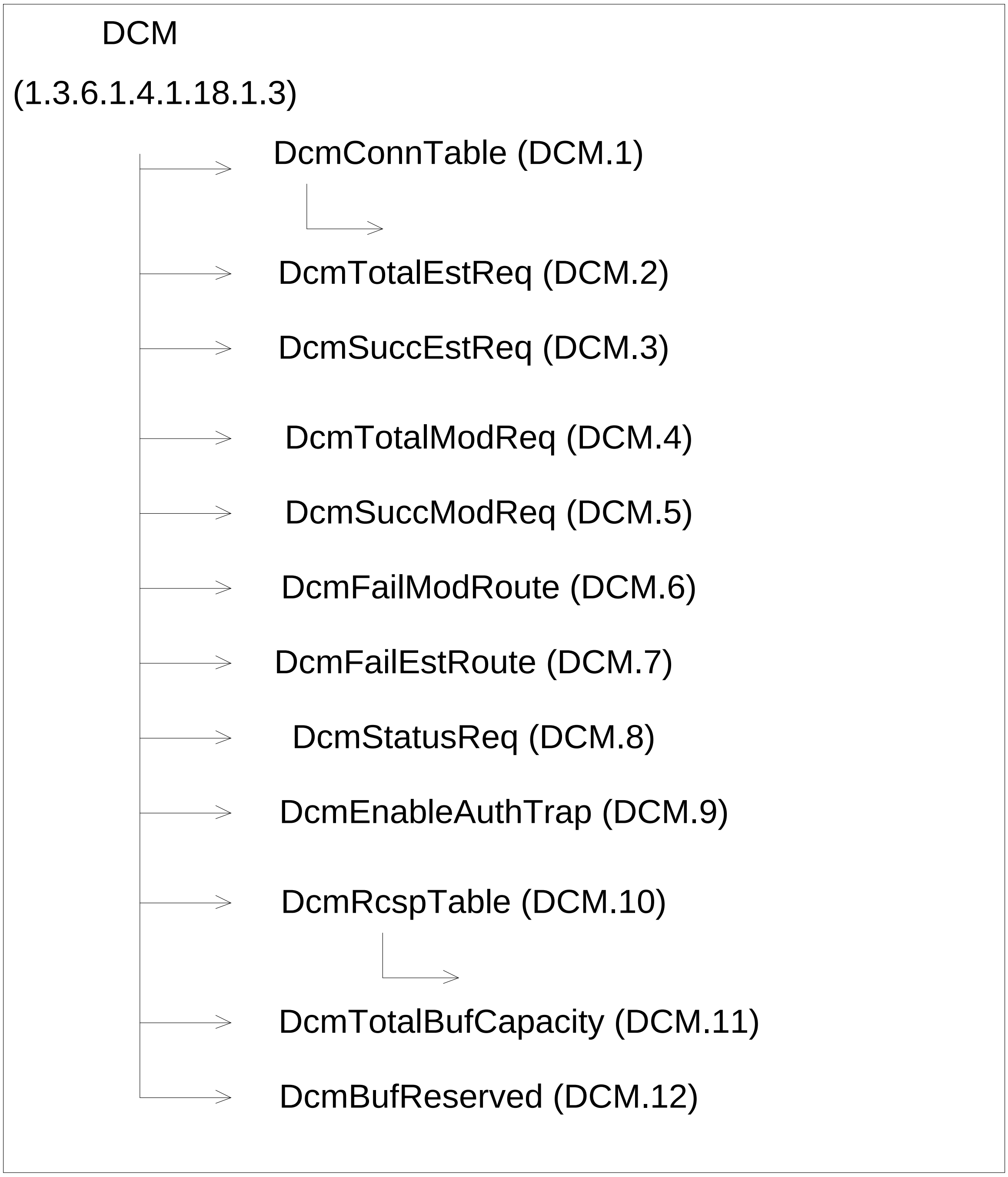

Figure 5.9: DCM Managed Objects

12. **DcmBufReserved:** The total buffer space currently reserved at this node.

The first table, **DcmConnTable**, contains the state of the real-time connection and is used to achieve channel control capability. The **SetRequest** and **GetResponse** PDU transferred between the DCM policy manager and the SNMNPv1 agent (as discussed in Section 5.3.2) contain the scalars comprising an entry in this table. The "setting" of this entry in the table initiates a control action that can establish, modify or terminate a real-time channel. The **DcmConnTable** is shown in Figure 5.10 and is described below.

The **DcmConnTable** is comprised of instances of the object **DcmConnEntry**. This object is a **SEQUENCE** of the objects as described below:

1. **DcmSourceChannelAddr:** The IP address of the source host. (index)
2. **DcmSourcePortNumber:** The port number of the source host. (index)
3. **DcmDestChannelAddr:** The IP address of the destination host. (index)
4. **DcmDestPortNumber:** The port number of the destination host. (index)
5. **DcmLcid:** The local channel identifier at this node.
6. **DcmXmin:** The Xmin value of the connection.
7. **DcmXave:** The Xave value of the connection.
8. **DcmI:** The Interval value of the connection.
9. **DcmSmax:** The Smax value of the connection.
10. **DcmDelay:** The end-to-end delay bound of the connection.
11. **DcmJitter:** The end-to-end delay jitter bound of the connection.
12. **DcmRoute:** A string containing the IP addresses of all nodes along the route.
13. **DcmStatus:** The status of the connection. The connection can be in the following states: establishing, modifying, terminating or established.
14. **DcmLocalDelay:** A string containing the local delay bounds at each node along the route of the connection.
15. **DcmRouteLcids:** A string containing the local channel identifiers at each node along the route.

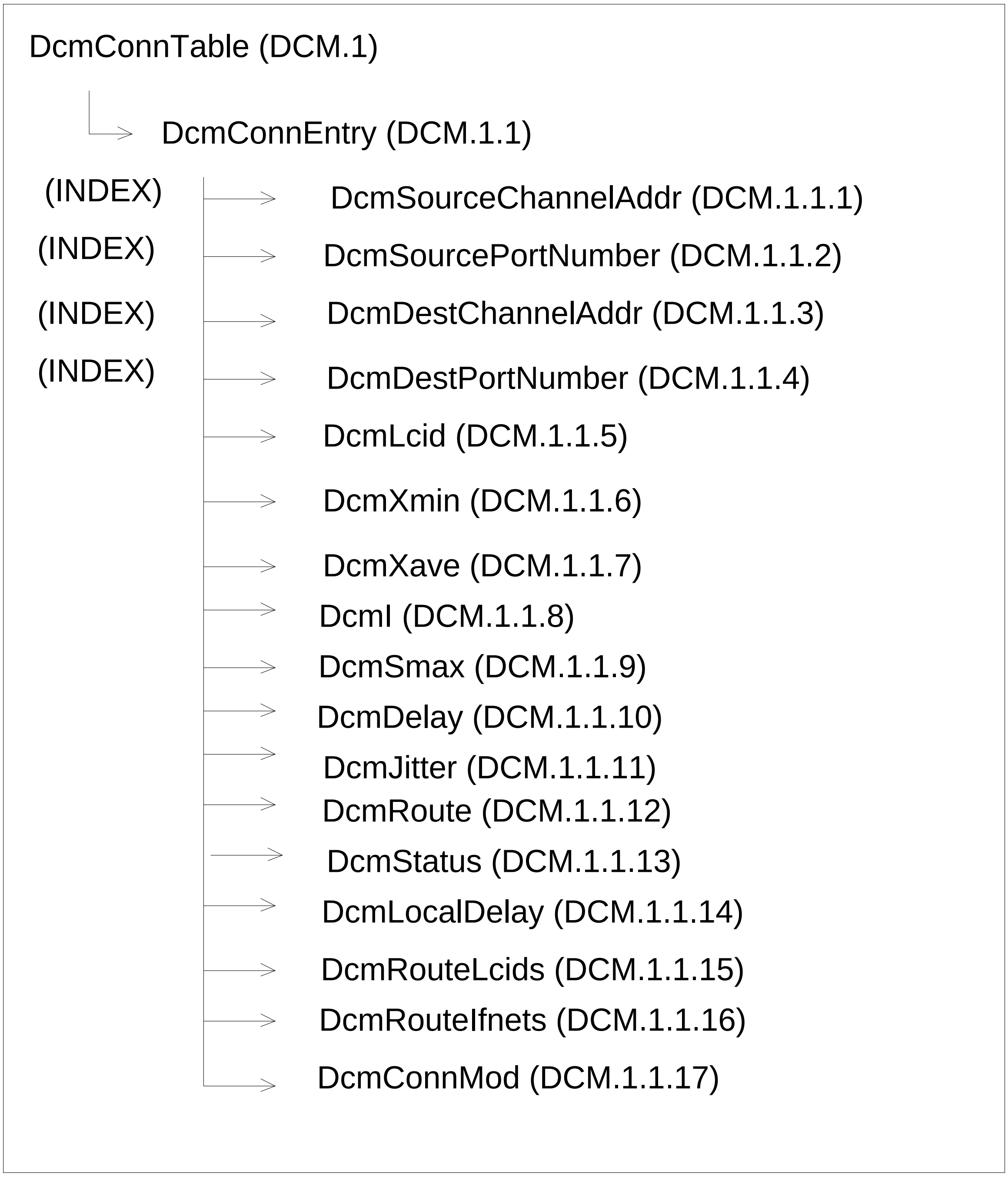

Figure 5.10: DcmConnTable Object

16. **DcmRouteIfnets:** A string containing the identifiers of the outgoing interfaces at each node along the route.

17. **DcmConnMod:** The number of times the channel has been modified.

The first 13 objects defined in the DcmConnTable object are read-write (RW), the remaining 4 read-only (RO). All objects are mandatory (M). Appendix I describes the details of these objects. The first 4 objects in the sequence comprise the INDEX for the table as they uniquely identify the channel. This table can be queried to obtain *static* information pertaining to any connection traversing the links supported by the node, while the **RtipConnTable** can be used to obtain the *dynamic* or runtime information pertaining to a connection at this node. The values of these 13 read-write objects are provided by the client or by the network; the DCM policy manager issues a **set** command to the SNMPv1 agent to accomplish the action indicated by the value of **DcmStatus** object[8]. This action can be the *establishing*, *modifying*, or *terminating* of a channel. The **DcmStatus** object indicates the current action desired by the client, and also if the channel is now established. When a channel is being established (i.e., on the forward pass), a new instance of the object is created in the **DcmConnTable** Table at each node using the requested traffic and performance parameters and the route provided from the **SetRequest** PDUs, and with the value of the **DcmStatus** object indicating *establishing*. The effect of this **set** command at the local agent causes a channel establishment or modification action to be taken. The agent parses the value of the **DcmRoute** object which contains the complete route to be used in this routing attempt to determine the outgoing link upon which the admission control tests are to be conducted. It then conducts the establishment tests on that outgoing link using the values of the traffic and performance objects. If the tests are successful, the agent updates the values of the **DcmLocalDelay**, **DcmRouteLcids** and **DcmRouteIfnets** objects to reflect the local delay bound, the lcid and outgoing link, respectively, of the connection at this node. It then sends a **GetResponse** PDU to the downstream node (as discussed in Section 5.3.2). If the admissions control tests fail, then this entry is removed from the **DcmConnTable** table, and a **GetResponse** PDU is sent to the upstream node to remove the resources reserved for this connection and the instance of the **DcmConnEntry** in the **DcmConnTable** table at that node. At the destination node, if the connection is accepted, the destination SNMPv1 agent calculates the new local delay bounds (i.e., the agent relaxes the delay bounds), and returns these values to the upstream nodes via the **DcmLocalDelay** object. On the reverse pass the delays are relaxed using the value of the **DcmLocalDelay** object. The value of the **DcmStatus** object is then changed to *established* and a **GetResponse** PDU is sent to the upstream node. In a channel modification the instance of the channel that is currently active is maintained with

[8]The value of an object usually indicates a *state*, but in SNMPv1, the manner in which an action is initiated is by writing a pre-specified value into an object.

the value of the **DcmStatus** object changed to *modifying* and the values of the traffic and performance parameter objects changed to reflect those of the modification request. If the modification request fails, the values of all previously changed objects are replaced with their original values. The value of the **DcmConnMod** object indicates the number of times that the connection has been modified.

The **DcmRcspTable** is comprised of instances of the object **DcmRcspEntry**, as shown in Figure 5.11. This object is a **SEQUENCE** of the objects described below.

1. **DcmRcspIndex:** The priority level in the queue.
2. **DcmRcspDelay:** The delay bound value at this priority level in the queue.
3. **DcmRcspValue:** The resources currently reserved at this level in the queue.

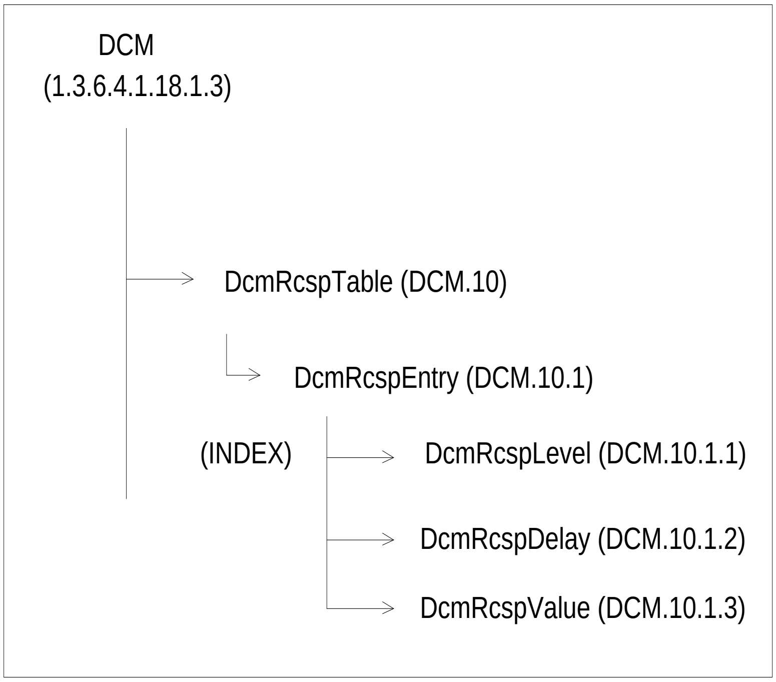

Figure 5.11: DcmRcspTable Object

5.4 Experiments

In this section we evaluate the performance of the DCM implementation on a local area testbed. This preliminary evaluation seeks to verify the correct functionality of the implementation and to obtain some initial performance measurements. Towards that end, we conducted two experiments that investigated the monitoring and control capabilities of the implementation. We begin this section by presenting the testbed environment followed by a description of the experiments, their results and analyses.

5.4.1 Testbed Environment

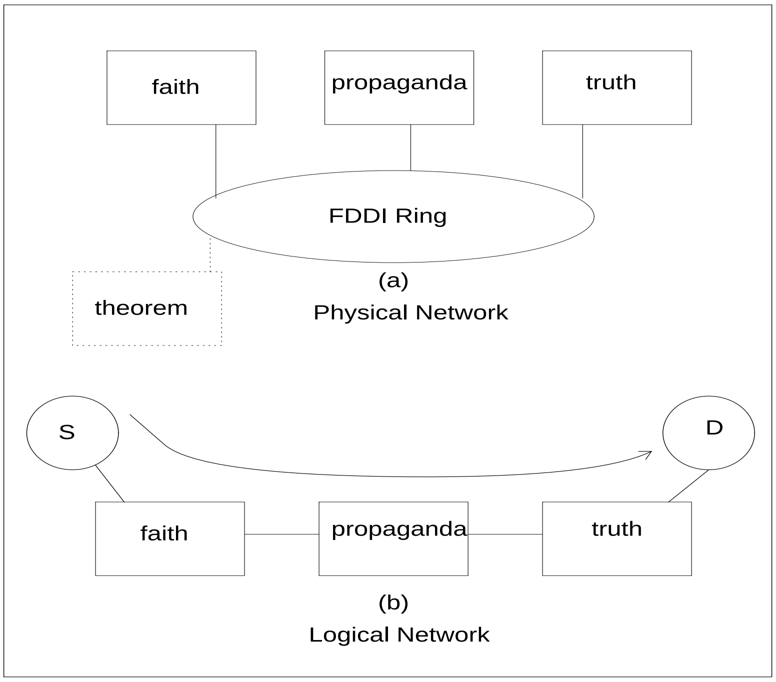

Figure 5.12: Local Area Testbed

The physical configuration of the local area testbed is presented in Figure 5.12(a). Of the four workstations depicted in the physical configuration three of them, namely *faith*, *propaganda*, and *truth*, are DECstation 5000/240s with the fourth being *theorem*, a DECstation 5000/125. All four workstations are connected to a FDDI ring with the routing algorithm (in the DCM scheme) adapted in such a manner that the three DECstation 5000/240s are configured to form the logical network shown in Figure 5.12(b). The DECstation 5000/125 is used to monitor the connections established across the logical network.

5.4.2 Monitoring Experiment

In order to verify the functionality of the monitoring capability of the implementation we conducted a simple experiment that *walked* a real-time connection and obtained various performance statistics from each node comprising the connection. This scenario is envisioned in the case where the network manager is attempting to determine the node responsible for *lost* or *late* packets on a connection. These statistics and their associated objects in the RTIP and RMTP groups were :

1. the number of packets sent, if at the source node, from this node (**RtipOutPkts**),

2. the number of packets forwarded, if at an intermediate node, at this node (**RtipForPkts**),

3. the number of packets received, if at an intermediate or destination node, at this node (**RtipInPkts**),

4. the number of packets that were late at this node (**RtipInLatePkts**),

5. the number of incoming messages at the destination node (**RmtpInMess**),

6. the number of outgoing messages at the source node (**RmtpOutMess**).

A real-time channel was established between a source client at *faith* and a destination client at *truth*, thus traversing the intermediate node *propaganda* (as shown in Figure 5.12(b)). The channel was established with the following parameters: $Xmin$ = 100 ms , $Xave$ = 100 ms, I = 1000 ms , $Smax$ = 1024 bits, D = 80 ms. Data was sent over the connection for 5 minutes, and the values of the objects mentioned above were obtained using the SNMPv1 agents at each node. The connection was monitored by an application program running on *theorem*, which executes a series of **get** commands[9] on the **DcmConnTable** object with the index values (**DcmSourceChannelAddr**, **DcmSourcePortNumber**, **DcmDestChannelAddr**, **DcmDestPortNumber**) = (128.32.33.105, 3769, 128.32.33.115, and 4001). The values of these indices uniquely identified the real-time channel. In order to *monitor* the entire connection, two values of the columnar objects **DcmRoute** and **DcmRouteLcids** are needed. These values provide the nodes comprising the route of the connection, in Internet dot notation, and the local channel identifier at each of these nodes. The value of the **DcmRoute** object is *128.32.33.105:128.32.33.107:128.32.33.115*, and the value of the **DcmRouteLcids** object is 1:1:1. With this **static** information, several **Get** commands were issued to each of the nodes to get the above statistics, with each node retrieving a different subset of the statistics. For instance, the source node retrieves statistics 1 and 6, the intermediate node statistics 2, 3, and 4, and the destination node statistics 3, 4, and 5. Tables 5.3 and 5.4 provide the results of these **get** commands and the times taken to retrieve and display these results.

As can be seen from the data in Table 5.3, there were no dropped or late packets on the connection, as the number of outgoing packets, **RtipOutPkts**, at the source and the number of incoming packets, **RtipInPkts**, at the destination are equal, and the number of late packets **RtipInLatePkts** at both the intermediate and destination nodes are zero. The results in Table 5.4 show the average and total time taken in querying the SNMPv1 agent. From this data, the average time per call was 24.07 ms, which was spent in SNMPv1 operations in the DEC Ultrix4.2a SNMPv1 libraries. These operations include the searching of the MIB tree to determine the validity and type of each object, the building of an SNMPv1 packet (using the ASN.1 notation), the sending of the packet, the receiving of the return

[9]In this instance SNMPv2 would have proven to be much more efficient, due to its capability to retrieve the values of several objects with its **GetBulkRequest** operation.

RMTP or RTIP Object	Source Node Value *(packets)*	Intermediate Node Value *(packets)*	Destination Node Value *(packets)*
1. RtipOutPkts	2998	-	-
2. RtipForPkts	-	2998	-
3. RtipInPkts	-	2998	2998
4. RtipInLatePkts	-	0	0
5. RmtpInMess	-	-	2998
6. RmtpOutMess	2998	-	-

Table 5.3: Monitoring Data

packet, and the parsing and verification of its contents. The average time taken to complete only these SNMPv1 operations was 20.2 ms. As the source code for the SNMPv1 libraries was not available, it was not possible to optimize this code; hence, it is currently our performance bottleneck. Thus, the total time taken to *monitor* this connection and collect its data is 240.95 ms (i.e., to the sum of the values of each of the calls in the **Command Group** in Table 5.4). This time includes the time to retrieve the *static* monitoring information, and to access each node, in turn, and retrieve the appropriate *runtime* performance statistics.

Command Group	Number of Gets	Average Time per call(ms)	Total Time per Group (ms)
Initial call to DCM group	2	24.1	48.2
Retrieve source node statistics	2	24.2	48.4
Retrieve inter. node statistics	3	24.0	72.0
Retrieve dest. node statistics	3	24.05	72.15

Table 5.4: Monitoring Times

The times presented in Table 5.4 do not reflect the time taken to display the statistics at the console[10]. It should be noted that to examine the *runtime* state of any single performance statistic of a channel, at any node, requires at most 3 **get** commands. The first 2 commands acquire the values of the **DcmRoute** and the **DcmRouteLcids** objects, thus identifying the appropriate instance in the **RtipConnTable** from which to obtain the relevant statistic (e.g., **RtipInPkts**). Therefore, the average time taken to obtain this performance measure is 72.27 ms. Once the route and lcid are obtained, then the relevant statistic can be obtained with a single **get** command with an average time of 24.09 ms. As the dynamics of interest operate in the seconds timescale, *runtime* monitoring of several statistics (in our case three

[10]These "printf" commands can take from 1.98 ms to 4.89 ms to execute.

statistics) can occur in a sufficiently short time, under 0.1 second, to permit a control action to be taken.

5.4.3 Control Experiment

The experiment that was used to accomplish the initial functional verification and analysis of the control capability of the implementation was a simple throughput modification experiment.

A real-time channel was established (as shown in Figure 5.12(b)) between a source client at *faith* and a destination client at *truth.* Two other background real-time channels were established and maintained during this experiment in addition to the usual production traffic on the FDDI ring[11]. The traffic characteristics and performance requirements of the initial and modified channel are provided in Table 5.5.

Parameters	**Initial Channel**	**Modified Channel**
Xmin (ms)	4	2
Xave (ms)	4	2
I (ms)	1000	1000
Smax (bytes)	1024	1024
D (ms)	80	80
Bandwidth (Mbps)	2.0	4.0

Table 5.5: Traffic and Performance Parameters

The duration of this experiment was 30 minutes, with channel modification occurring 15 minutes after the starting time. The modification contract used was a *No-Violation* modification contract. However, as there was no change in route (due to the simple topology of our testbed), we did not expect nor experience any out-of-sequence packets. Figure 5.13 provides a throughput graph resulting from the experiment, with the channel's throughput over the last second computed at each second.

As shown in this figure, at time 15 minutes the throughput of the channel is modified from 2.0 Mbps to 4.0 Mbps. The monitoring capability of the implementation was used to verify that there were no *late* packets (i.e., packets that exceeded their delay bounds) or *out-of-sequence* packets. The average execution times of establishment and modification actions on a single pass at the intermediate node *propaganda* are shown in Table 5.6. These execution times represent the time interval between the receipt of the request by the DCM policy manager at the source and the return of the policy manager's response on a single pass in either the forward or reverse direction[12]. These averages were calculated

[11]These workstations are used as general purpose machines by other members of the Tenet Group.

[12]They do not account for the time spent in the *Client/Manager* socket-based interface.

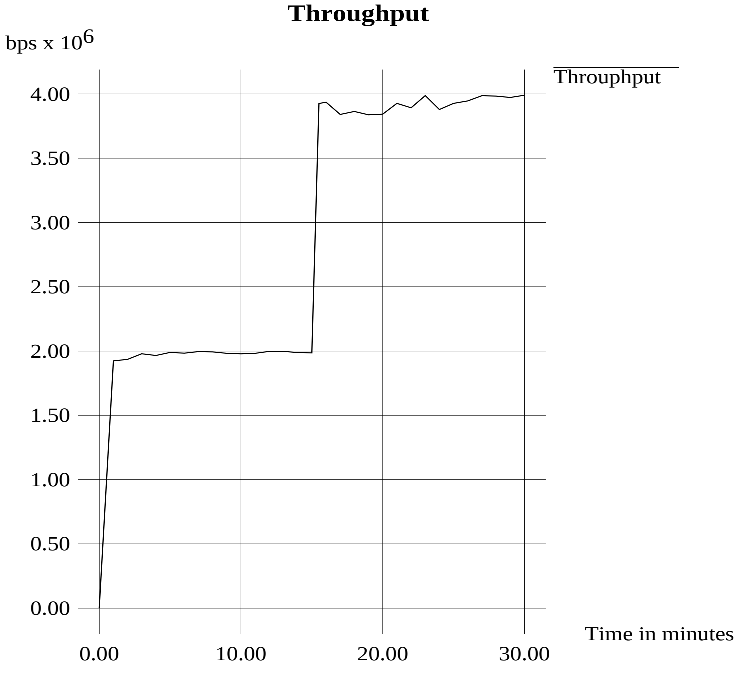

Figure 5.13: Throughput of Channel

over 50 experimental runs so as to remove any effects due to temporary fluctuations in the cpu or network load; thus, these average execution times are averaged over the execution times on each pass (be it forward or reverse) for all of the experimental runs. The execution times at this node are nearly identical to those experienced at the other two nodes, *faith* and *truth*. These establishment times are analyzed in Table 5.6, where the time spent on SNMPv1 operations (which could be optimized) is distinguished from the time used by DCM processing and the admission control tests.

Procedure	**Establishment Message (ms)**	**Modification Message (ms)**
Initializing SNMP data structures	10.1	10.2
Preparing the SNMP SetRequest PDU	2.4	2.4
Sending a reliable SetRequest PDU	7.8	7.7
Processing the DCM message	1.2	1.2
Executing the DCM admission tests	0.05	0.06
Total time at each node	21.55	21.56

Table 5.6: Average Execution Times at the Intermediate Node

The first procedure, *Initializing the SNMP data structures*, initializes the SNMP data structures with information such as the community name (i.e., the access permissions), the number of retries, the timeout value, etc[13]. The *Preparing the SNMP SetRequest PDU* procedure parses the DCM data structure that is given to it, retrieves the MIB object identifiers, and creates the protocol data unit in ASN.1 notation. The *Sending a reliable SetRequest PDU* procedure is responsible for sending the SetRequest PDU and receiving the GetResponse PDU from the agent. *Processing the DCM message* is responsible for examining the request and, based on the DCM policies, executing the appropriate action, which may result in sending a PDU to the local agent or in responding to a source or destination client. Table 5.7 presents the average times taken to establish and modify the channel over 50 experimental runs. These times reflect both the forward and reverse pass across the three nodes traversed by the connection.

Procedure	**Establishment Message (ms)**	**Modification Message (ms)**
Synchronizing the agent	60.6	61.2
Preparing the snmp SetRequest PDU	14.4	14.4
Sending a reliable SetRequest PDU	46.8	46.2
Processing DCM message	7.2	7.2
DCM admission tests	0.3	0.36
Total time	129.3	129.36
Percentage time in SNMP operations	94.2%	94.16%
Percentage time in DCM operations	5.8%	5.84%

Table 5.7: Average Establishment and Modification Times of the Connection

As can be seen in Table 5.7, the average channel establishment and modification times across this route are approximately 129.3 ms, 94.2% of which is time spent in SNMPv1 operations. The propagation delay on this local testbed was not significant. Given the timescale of interest to us, this establishment time is within acceptable bounds. As shown in Table 5.6, each node executes its establishment or modification operations, on average, in 21.5 ms. In our three-node experiments, each node is "touched" twice (once on the forward pass and once on the reverse pass), thus there are 6 "touches" during then establishment or modification of a channel, which would require a total operation time, on average, of 129 (i.e., 21.5 * 6) ms. It should be noted that this corresponds approximately to the average establishment and modification times of 129.3 ms recorded in the three-node experiment (see Table 5.7).

The use of a WAN instead of a LAN would add the cost of additional hops and the propagation delays of the additional links, which, in the case of a cross country connection (using the XUNET topology as an example, there are 6 hops between Berkeley and Bell Laboratories, Murray Hill and the round

[13] It utilizes several commands in the SNMP library which are *expensive*.

trip propagation delay is 50 ms), would result in an establishment time of approximately 306 ms; this would still be acceptable in the seconds timescale. It should be noted that our SNMPv1 code is not optimized for speed; discussions with its implementors suggest that a 40% reduction in execution time is easily possible[14]. This reduction would place the establishment times of at approximately 80.5 ms, with a WAN connection (mentioned above) being established or modified in a very acceptable 209.6 ms.

5.5 Summary

In this chapter we presented an implementation of the DCM scheme in a local area testbed and two experiments design to provide functional verification of the prototype and to examine the suitability of SNMPv1 as a management protocol.

The scheme was implemented, using the Simple Network Management Protocol (SNMPv1), on a DEC platform under Ultrix4.2a, and provides both monitoring and control capabilities to the network. The guaranteed-performance connections (which use the Real-Time Message Transport Protocol (RMTP) and the Real-Time Internet Protocol (RTIP)) can be monitored and controlled by SNMPv1 agents under the guidance of DCM policy managers. Monitoring and control actions are achieved by using the SNMPv1 **get**, **set** and **trap** operations on the RTIP, RMTP, and DCM Management Information Bases (MIBs). **Get** operations retrieve the *static* and *runtime* data pertaining to a connection using all three MIBs. Connection control is achieved by doing a **set** operation on the values of specified objects in the DCM MIB. The "setting" of these objects initiates the establishment, modification, and termination actions using the values that have been provided in the objects corresponding to the traffic and performance parameters.

The monitoring and control experiments were conducted on a local area testbed comprised of four DECstations connected by a FDDI ring. Our measurement studies verified that the implemention did achieve the intended functionality; however, they showed the performance limitations due to the DEC implementation of the SNMPv1 agent. In the monitoring experiments the total time taken to retrieve any single statistic of a real-time channel is on the average 24.07 ms (i.e., the time taken to execute a SNMPv1 **get** command and receive the results); this time does not include the initial call to the DCM group to retrieve the *static* information (i.e., the route traversed by the connection and the lcids at each of the nodes comprising this route). The total time taken to obtain the values of several statistics, that is, to retrieve the *static* information and a number of *runtime* statistics (e.g., three statistics), would take on the average 122.3 ms, which is acceptable in our timescale of interest. In our control experiments, the average execution time of establishment and modification actions at a node (on a single pass in either direction) is 21.55 ms. This time is dominated by inefficient SNMPv1 library operations

[14]This is a conservative estimate based on some initial profiling that was done on the library functions.

which are responsible for 94.2% of the time. The average time taken to establish or modify a three-node connection in our local testbed was 129.3 ms, which is acceptable in our timescale; however, 121.5 ms of this time was due to SNMPv1 operations. As this SNMPv1 code can be optimized to reflect a reduction in execution time of at least 40%, the average modification time in a three-node connection can be reduced to approximately 80 ms.

Chapter 6

Summary and Future Work

6.1 Summary

In this chapter we summarize the main points of this dissertation and we discuss some topics for future work. The main goal of this dissertation was to increase the *flexibility* of Guaranteed Performance Connection (GPC) services by permitting them to adapt to the dynamics of client demands and network state. Our method of achieving this goal was by *proof of concept*, that is, we designed the Dynamic Connection Management (DCM) scheme, which is a collection of algorithms and mechanisms that would permit such adaptation, we analyzed the behavior of DCM through simulation experiments, and we implemented a prototype of the scheme.

In Chapter 1, we presented the new communication requirements that arise from the current user demands for total information access, uniform connectivity, interactive communication, and the desire for multiple media type presentations. The attempts to satisfy these demands in an efficient manner have resulted in a number of proposed designs for *packet-switched integrated services networks*. These networks seek to provide a wide range of qualities of service to users and to utilize statistical multiplexing to increase resource utilization. The qualities of service they provide to the users must include guarantees on various performance indices for a client's traffic with given characteristics. These guaranteed-performance services have been called GPC services. The efficient support of multiple qualities of service constitutes one level of *adaptation* or *flexibility* to user demands. In that chapter, we illustrated another level of *adaptation* or *flexibility* that was desired by the users. At this level, the dynamics of client demands and of the network state require that the network adapt the service it offers to clients during the lifetimes of the clients' communication sessions. Several examples, such as performance parameter tuning, media scaling, load balancing, were provided to illustrate the dynamics of interest to us. These examples demonstrate the need for dynamic GPC services, which permit modification of the traffic characteristics, the performance parameters and the routes of the connections during their lifetimes.

In Chapter 2, we provided a review of the relevant literature and a critical analysis of related research. This review was done in two sections. In the first section, we examined the relevant GPC schemes, and, in the second, routing under performance constraints. The GPC schemes were reviewed to determine their strengths and weaknesses. In order to provide *efficient* flexibility, the choice of a connection's route is important. Thus, the examination of routing techniques that consider performance constraints was necessary. Both circuit- and packet-switching routing techniques were reviewed to determine their suitability for use in our work.

In Chapter 3, we proposed a dynamic resource management scheme, called the Dynamic Connection Management (DCM) scheme, which permits the modification of the traffic characteristics, the performance parameters, and the route of a connection subject to a modification contract. This modification contract specifies the degree of disruption that a connection can experience during modification. The degree of disruption can range from no performance violations to a bounded number of performance violations. The DCM scheme is an enhancement of the Tenet GPC scheme and is a collection of three algorithms: the DCM channel administration algorithm, the DCM routing algorithm, and the DCM transition algorithm. The DCM channel administration algorithm reserves the network resources, in the presence or absence of resource sharing, needed to support the transition from the original (i.e., *primary*) channel to the new (i.e., *alternate*) channel and to ensure that the performance guarantees of the alternate channel are satisfied. The DCM routing algorithm determines a route from the source to the destination host based on the traffic and performance requirements of the connection and the resource sharing factor. The DCM transition algorithm ensures that the performance violations specified in the DCM modification contract are adhered to during the transition. These three algorithms were described in detail in this chapter. The DCM scheme also supports mechanisms that enable modifications to a connection to be made to a segment of the connection (*local control*) or to the entire connection (*global control*). Faster establishment and modification is also possible as the DCM scheme utilizes the *intelligent restart* establishment procedure, which is based on the time value of the network state information and permits the bypassing of resource saturated links during establishment.

In Chapter 4, we described the simulator that was built to analyze the DCM scheme, and we presented and analyzed the results of several simulations experiments which were conducted to determine the validity and performance of the scheme. In order to correctly analyze the scheme, we proposed a new metric, the *Queuing Delay Index*, that captures both bandwidth and delay resources comprising the network load in a single value, thus permitting us to categorize a network's load or to compare the resources consumed by two or more real-time channels. Our experiments verified the functionality of the scheme in that all traffic, performance and route modifications were realized within the constraints of the modification contracts. The experiments also showed that, under our workload and topological conditions, establishment and modifications times were fully adequate for our timescales of interest. The effectiveness of the *intelligent restart* procedure was also examined and compared to that of the conventional Tenet establishment procedure. Our analysis showed that, at *low* and *medium* network loads, the *intelligent restart* procedure establishes (or modifies) a channel faster than the conventional establishment procedure.

In Chapter 5, we concluded our *proof of concept* by providing a prototype implementation of the DCM scheme, and presenting the results of initial experiments conducted on this prototype. In our implementation we attempted to provide the basic management mechanisms, using a standardized “open” management framework, namely SNMPv1, by which guaranteed-performance connections can be monitored and controlled. SNMPv1's simplicity, availability, and wide dissemination justify its use over that of a more complex, though more complete, management protocol. The results of these initial experiments indicated that the implementation was functionally correct. Our performance measurements indicated that both connection monitoring and control capabilities performed within acceptable times for our timescale of interest. An analysis of these measurements also indicated that the execution times of the Ultrix4.2a SNMPv1 library routines occupied a significant portion of the average establishment time (in the case of our local area testbed, SNMPv1 library routines occupied 92.4% of the establishment time). These library routines are known to be inefficient; hence, if the routines were made more efficient then the execution times at each node (and channel establishment times) would be even shorter. A conservative estimate on the possible reduction of these SNMPv1 operational times was given as 40 %, which would make the performance times of monitoring and control operations even more acceptable in our timescale of interest.

6.2 Future Work

While this thesis has offered an initial solution to the problem of the adaptability of guaranteed-performance communication services, a number of issues remain to be explored.

The DCM scheme solves the problem of the flexibility of a “deterministic” real-time channel, but does not address that of flexible “statistical” channels. A solution to this problem would involve the re-design of the algorithms and also the redefinition of the modification contracts.

The DCM scheme was analyzed using synthetic workloads during our simulation experiments. We would like to perform some additional analysis using traces from “real” workloads, on the topology of a production network, to gain a more useful measure of the scheme's suitability.

An analysis of the effects of the interactions between the DCM scheme and the DCM policies needs to be undertaken. The DCM policies determine the rate at which modifications are allowed, the modification parameters, and the degree of control that is permitted (i.e., if local or global modifications are to be done). As the DCM scheme operates in finite time, the rate, the values of the parameters, and the degree of control of channel modification must be considered so that network utilization and establishment times are not adversely affected.

Our implementation of the DCM scheme needs to be optimized by modifying the inefficient routines in the SNMPv1 library. A better alternative to this is the use of SNMPv2 with an efficient library. More complete MIBs need to be defined to support the higher level management functions needed, such as fault management of real-time channels (this area is currently being investigated within the Tenet Group).

Finally, the implementation needs to be extended to an internetwork environment and investigated using non-synthetic workloads. These studies would be more fruitful using the realistic workloads of a "production-like" environment.

The above list is by no means an exhaustive list of the work that needs to be done in this area; rather, it is a short compilation of our wishes. As always, with the further experiences gained in these explorations, we will undoubtedly encounter new topics for research.

Appendix A

Appendix I presents the MIB for the RTIP, RMTP, and DCM groups.

```
--   the RTIP group


--   Implementation of this group is mandatory . It is used to support
--   guaranteed performance connections in the Tenet framework.


RtipVerNum  OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
           " The version number of RTIP used at this node."
    ::= { Rtip 1 }

RtipMaxConn  OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
           " The maximum number of RTIP connections permitted
     at this node."
    ::= { Rtip 2 }

RtipNumConn  OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
           " The current number of RTIP connections at this node."
    ::= { Rtip 4 }


RtipConnTable  OBJECT-TYPE
```

```
    SYNTAX  SEQUENCE OF RtipEntry
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
           " A list of the Rtip connection entries."
    ::= { Rtip 3 }


RtipEntry OBJECT-TYPE
    SYNTAX  RtipEntry
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
    " An Rtip connection entry containing
packet statistics for a connection"
    INDEX   { RtipLcid, RtipLocalChannelAddr }
    ::= { RtipConnTable 1 }

 RtipEntry ::=
SEQUENCE {
   RtipLcid
INTEGER,
   RtipLocalChannelAddr
IpAddress,
   RtipConnStat
INTEGER,
     RtipInPkts
Counter,
   RtipInLatePkts
Counter,
   RtipInHddrErr
Counter,
   RtipInChkErr
Counter,
```

```
    RtipOosPkts
Counter,
    RtipForPkts
Counter,
    RtipOutPkts
Counter,
    RtipDownLcid
INTEGER,
    RtipDownChannelAddr
IpAddress,
}

   RtipLcid  OBJECT-TYPE
      SYNTAX  INTEGER
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
             " A unique value for each connection. This is the local
       channel identifier (lcid) for the connection."
      ::= { RtipEntry  1 }

   RtipLocalChannelAddr  OBJECT-TYPE
      SYNTAX  IpAddress
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
             " The source ip address of the connection."
      ::= { RtipEntry  2 }

    RtipConnStat  OBJECT-TYPE
      SYNTAX   INTEGER
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
```

```
            " The status of the node (source, intermediate,or
destination node). "
      ::= { RtipEntry 3 }

    RtipInPkts  OBJECT-TYPE
      SYNTAX  Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
             " The number of packets that have been received
by this connection."
      ::= { RtipEntry 4 }

    RtipInLatePkts  OBJECT-TYPE
      SYNTAX  Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
             " The number of packets that have been received late at this node
on this connection."
      ::= { RtipEntry 5 }

    RtipInHddrErr  OBJECT-TYPE
      SYNTAX  Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
             " The number of incoming packets received at this node
with header errors."
      ::= { RtipEntry 6 }

    RtipOosPkts  OBJECT-TYPE
      SYNTAX  Counter
      ACCESS  read-only
```

```
      STATUS  mandatory
      DESCRIPTION
             " The number of packets that have been received out of sequence
at this node on this connection."
      ::= { RtipEntry 7 }

    RtipInChkErr  OBJECT-TYPE
      SYNTAX  Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
             " The number of packets that have been received with bad CRCs
at this node on this connection."
      ::= { RtipEntry 8 }

    RtipForPkts  OBJECT-TYPE
      SYNTAX  Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
             " The number of packets that have been forwarded
at this node on this connection."
      ::= { RtipEntry 9 }

    RtipOutPkts  OBJECT-TYPE
      SYNTAX  Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
             " The number of packets that have been sent on this
connection (i.e. with this node as the source node)."
      ::= { RtipEntry 10 }

   RtipDownLcid  OBJECT-TYPE
```

```
   SYNTAX  INTEGER
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
          " The local channel identifier (lcid) at the downstream node
    in this connection."
   ::= { RtipEntry  11 }

 RtipLocalChannelAddr  OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
          " The next downstream ip address of the connection."
   ::= { RtipEntry  12 }



--   the RMTP group

--   Implementation of this group is mandatory . It is used to support
--   guaranteed performance connections in the Tenet framework in the
--   Real-Time Message Transport Protocol (RMTP).


RmtpMaxConn  OBJECT-TYPE
   SYNTAX  INTEGER
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
          " The maximum number of RMTP connections permitted
    at this node."
   ::= { Rmtp 1 }
```

```
  RmtpNumConn  OBJECT-TYPE
      SYNTAX  Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
             " The current number of RMTP connections at this node."
      ::= { Rmtp 2 }



  RmtpConnTable  OBJECT-TYPE
      SYNTAX  SEQUENCE OF RmtpEntry
      ACCESS  not-accessible
      STATUS  mandatory
      DESCRIPTION
             " A list of the Rmtp connection entries."
      ::= { Rmtp 3 }


  RmtpEntry OBJECT-TYPE
      SYNTAX  RmtpEntry
      ACCESS  not-accessible
      STATUS  mandatory
      DESCRIPTION
      " An Rmtp connection entry containing
message statistics for a connection"
      INDEX   { RmtpSourceLcid, RmtpSourceChannelAddr }
      ::= { RmtpConnTable 1 }

   RmtpEntry ::=
SEQUENCE {
    RmtpSourceLcid
INTEGER,
    RmtpSourceChannelAddr
```

```
IpAddress,
    RmtpInMess
Counter,
    RmtpInReassErr
Counter,
    RmtpOutMess
Counter,
    RmtpOutErr
Counter,
}

   RmtpSourceLcid  OBJECT-TYPE
      SYNTAX  INTEGER
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
             " A unique value for each connection. This is the local
       channel identifier (lcid) for the connection."
      ::= { RmtpEntry  1 }

    RmtpSourceChannelAddr  OBJECT-TYPE
      SYNTAX   IpAddress
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
             " The source ip address of the Rmtp connection."
      ::= { RmtpEntry 2 }

    RmtpInMess  OBJECT-TYPE
      SYNTAX   Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
             " The number of messages received on connection."
```

```
      ::= { RmtpEntry 3 }

    RmtpInReassErr  OBJECT-TYPE
      SYNTAX   Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
             " The number of reassembly errors that occurred in
processing the incoming messages."
      ::= { RmtpEntry 4 }

    RmtpOutMess  OBJECT-TYPE
      SYNTAX   Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
             " The number of outgoing messages on this connection
(i.e.  the number of messages sent on the connection)."
      ::= { RmtpEntry 5 }

    RmtpOutErr  OBJECT-TYPE
      SYNTAX   Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
             " The number of outgoing messages that could not be sent
due to fragmentation errors."
      ::= { RmtpEntry 6 }



  --   the DCM group

  --   Implementation of this group is mandatory . It is used to support
```

```
--   guaranteed performance connections in the Tenet framework in the
--   Real-Time Message Transport Protocol (RMTP). This is the control
--   group.


DcmConnTable  OBJECT-TYPE
    SYNTAX  SEQUENCE OF DcmEntry
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
           " A row of the Dcmconnection entries."
    ::= { Dcm 1 }


DcmEntry OBJECT-TYPE
    SYNTAX  DcmEntry
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
    " A Dcmconnection entry containing traffic parameters
and performance requirements of a connection"
    INDEX   { DcmSourceChannelAddr, DcmSourcePortNumber, DcmDestChannelAddr, DcmDestPortNumber
    ::= { DcmConnTable 1 }

   DcmEntry ::=
SEQUENCE {
            DcmSourceChannelAddr
IpAddress,
    DcmSourcePortNumber
      INTEGER,
    DcmDestChannelAddr
IpAddress,
    DcmDestPortNumber
       INTEGER,
```

```
    DcmLcid
INTEGER,
    DcmXmin
INTEGER,
    DcmXave
INTEGER,
    DcmI
INTEGER,
    DcmSmax
INTEGER,
    DcmDelay
INTEGER,
    DcmJitter
INTEGER,
    DcmRoute
OCTET STRING,
    DcmStatus
INTEGER,
    DcmLocalDelay
OCTET STRING,
    DcmRouteLcids
OCTET STRING,
    DcmRouteIfnets
OCTET STRING,
    DcmConnMod
Counter,
}


  DcmSourceChannelAddr  OBJECT-TYPE
      SYNTAX  IpAddress
      ACCESS  read-write
      STATUS  mandatory
      DESCRIPTION
```

```
        " The source ip address of the connection."
    ::= { DcmEntry 1 }

DcmSourcePortNumber  OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
        " The source port number of the connection."
    ::= { DcmEntry 2 }

DcmDestChannelAddr  OBJECT-TYPE
    SYNTAX  IpAddress
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
        " The destination ip address of the connection."
    ::= { DcmEntry 3 }

DcmDestPortNumber  OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
        " The destination port of the connection."
    ::= { DcmEntry 4 }

DcmLcid  OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
        " Local Channel identifier for connection."
    ::= { DcmEntry 5 }
```

```
  DcmXmin  OBJECT-TYPE
      SYNTAX  INTEGER
      ACCESS  read-write
      STATUS  mandatory
      DESCRIPTION
             " Xmin - Minimum packet interarrival time. (In units of
       tenths of ms)"
      ::= { DcmEntry 6 }

  DcmXave  OBJECT-TYPE
      SYNTAX  INTEGER
      ACCESS  read-write
      STATUS  mandatory
      DESCRIPTION
             " Xave - Average packet interarrival time. (In units of
tenths of ms)."
      ::= { DcmEntry 7 }

  DcmI  OBJECT-TYPE
      SYNTAX  INTEGER
      ACCESS  read-write
      STATUS  mandatory
      DESCRIPTION
             " Interval over which the average is taken. (Units 1/10 ms)"
      ::= { DcmEntry 8 }

  DcmSmax  OBJECT-TYPE
      SYNTAX  INTEGER
      ACCESS  read-write
      STATUS  mandatory
      DESCRIPTION
             " Smax - Maximum packet size. (In bytes)"
      ::= { DcmEntry 9 }
```

```
  DcmDelay  OBJECT-TYPE
      SYNTAX  INTEGER
      ACCESS  read-write
      STATUS  mandatory
      DESCRIPTION
             " Delay bound required by client. (In units of
tenths of ms)."
      ::= { DcmEntry 10 }

  DcmJitter  OBJECT-TYPE
      SYNTAX  INTEGER
      ACCESS  read-write
      STATUS  mandatory
      DESCRIPTION
             " Jitter bound required by client. A value of zero
is interpreted as no bound needed.  (Units 1/10 ms)"
      ::= { DcmEntry 11 }

  DcmRoute  OBJECT-TYPE
      SYNTAX  OCTET STRING
      ACCESS  read-write
      STATUS  mandatory
      DESCRIPTION
             " Route from source to destination."
      ::= { DcmEntry 12 }

  DcmStatus  OBJECT-TYPE
      SYNTAX  INTEGER
      ACCESS  read-write
      STATUS  mandatory
      DESCRIPTION
             " State of connection. Connection can be establishing,
established, modifying, or terminating."
```

```
     ::= { DcmEntry 13 }

 DcmLocalDelay  OBJECT-TYPE
     SYNTAX  OCTET STRING
     ACCESS  read-write
     STATUS  mandatory
     DESCRIPTION
            "A string containing the local delay values along the route."
     ::= { DcmEntry 14 }

 DcmRouteLcids  OBJECT-TYPE
     SYNTAX  OCTET STRING
     ACCESS  read-write
     STATUS  mandatory
     DESCRIPTION
            "A string containing the local lcid values at each node
along the route."
     ::= { DcmEntry 15 }

 DcmRouteIfnets  OBJECT-TYPE
     SYNTAX  OCTET STRING
     ACCESS  read-write
     STATUS  mandatory
     DESCRIPTION
            "A string containing the outgoing interfaces at each node
along the route."
     ::= { DcmEntry 16 }

 DcmConnMod  OBJECT-TYPE
     SYNTAX  Counter
     ACCESS  read-only
     STATUS  mandatory
     DESCRIPTION
            " Number of times that connection has been modified."
```

```
    ::= { DcmEntry 17 }

DcmTotalEstReq  OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
           "Number of Establishment requests received."
    ::= { Dcm 2 }

DcmSuccEstConn  OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
           "Number of establishment requests that were accepted."
    ::= { Dcm 3 }

DcmTotalModReq  OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
           "Number of Modification requests received."
    ::= { Dcm 4 }

DcmSuccModReq  OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
           "Number of modification requests that were accepted."
    ::= { Dcm 5 }
```

```
  DcmFailModRoute  OBJECT-TYPE
      SYNTAX  Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
             " Number of modification requests failed due to
route unavailability."
      ::= { Dcm 6 }

  DcmFailEstRoute  OBJECT-TYPE
      SYNTAX  Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
             "Number of establishment requests failed due to
route unavailability. "
      ::= { Dcm 7 }


  DcmStatReq  OBJECT-TYPE
      SYNTAX  Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
             "Number of connection status requests received."
      ::= { Dcm 8 }


  DcmEnableAuthTrap  OBJECT-TYPE
      SYNTAX  INTEGER { enabled(1), disabled(2) }
      ACCESS  read-write
      STATUS  mandatory
      DESCRIPTION
             "Indicates whether the SNMP agent process is
```

```
      permitted to generate Dcmauthentication
      failure traps."
      ::= { Dcm 9 }


  DcmRcspTable  OBJECT-TYPE
      SYNTAX  SEQUENCE OF DcmRcspEntry
      ACCESS  not-accessible
      STATUS  mandatory
      DESCRIPTION
             " Table of Rate Controlled Static Priority state variables."
      ::= { Dcm 10 }


  DcmRcspEntry OBJECT-TYPE
      SYNTAX  DcmRcspEntry
      ACCESS  not-accessible
      STATUS  mandatory
      DESCRIPTION
      " An entry containing the values at each priority level."
      INDEX   { DcmRcspLevel }
      ::= { DcmRcspTable 1 }

   DcmRcspEntry ::=
SEQUENCE {
            DcmRcspLevel
INTEGER,
            DcmRcspDelay
INTEGER,
            DcmRcspValue
Counter,
}
```

```
DcmRcspLevel  OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
           "RCSP Priority number at this level."
    ::= { DcmRcspLevel 1 }


DcmRcspDelay  OBJECT-TYPE
    SYNTAX  INTEGER
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
           "Delay value at this RCSP level."
    ::= { DcmRcspLevel 2 }


DcmRcspValue  OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
           "Resources reserved at this RCSP level."
    ::= { DcmRcspLevel 3 }


DcmTotalBufCapacity  OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
```

```
        "Total buffer capacity at node."
    ::= { Dcm 11 }


DcmBufReserved  OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
        "Total buffers currently reserved at node."
    ::= { Dcm 12 }
```

Bibliography

[1] H. Ahmadi, J. Chen, and R. Guerin. Dynamic routing and call control in high-speed integrated networks. Technical report, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY., November 1991.

[2] H. Ahmadi and W. Denzel. Survey of modern high performance switching techniques. *IEEE Journal on Selected Areas in Communications*, 7(7):1091–1103, September 1989.

[3] David P. Anderson, Ralf Guido Herrtwich, and Carl Schaefer. SRP: A resource reservation protocol for guaranteed performance communication in internet. Technical Report TR-90-006, International Computer Science Institute, Berkeley, California, February 1990.

[4] G. Ash. Use of a trunk status map for real-time dnhr. In *International TeleTraffic Congress ITC-11*, 1985.

[5] G. Ash. Design and control of networks with dynamic nonhierarchical routing. *IEEE Communications Magazine*, October 1990.

[6] G. Ash, J. Chen, A. Frey, and B. Huang. Real-time network routing in a dynamic class-of-service network. In *International TeleTraffic Congress ITC-13*, 1991.

[7] A. Banerjea, D. Ferrari, B. Mah, M. Moran, D. Verma, and H. Zhang. *The Tenet Real-Time Protocol Suite: Design, Implementation, and Experiences.* unpublished draft, 1994.

[8] Anindo Banerjea and Bruce Mah. The real-time channel administration protocol. In *Proceedings of Second Int'l. Workshop on Network and Operating Systems Support for Digital Audio and Video*, Heidelberg, Germany, November 1991.

[9] Anindo Banerjea, Colin Parris, and Domenico Ferrari. Recovering guaranteed performance service connections from single and multiple faults. In *Proceedings of GLOBECOM '94*, San Francisco, CA, November 1994.

[10] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, pages 87–90, 1958.

[11] W. Cameron, P. Galloy, and W. Graham. Report on the toronto advance routing concept trial. In *Proceedings of the First International Planning Symposium*, France, Paris, 1980.

[12] CCITT proposed recommendation i.311, June 1991.

[13] Israel Cidon, Inder Gopal, and Roch Guerin. Bandwidth management and congestion control in PlaNET. *IEEE Communications Magazine*, pages 54–64, October 1991.

[14] D. Clark, S. Shenker, and L. Zhang. Supporting real-time applications in an integrated services packet network: Architecture and mechanism. In *Proceeding of the SIGCOMM'92*, pages 14–26, August 1992.

[15] 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.

[16] R. Cochi, D. Estrin, S. Shenker, and L. Zhang. A study of priority pricing in multiple service class networks. In *Proceeding of the SIGCOMM'91*, September 1991.

[17] D.C. Darragh and R.L. Baker. Fixed distortion subband coding of images for packet-switched networks. *IEEE Journal on Selected Areas in Communications*, 7(5):826–832, June 1989.

[18] L. Delgrossi, C. Halstrick, D. Hehmann, R.G. Herrtwich, O. Krone, J. Sandvoss, and C. Vogt. Media scaling with heits. In *Proceeding of ACM Multimedia '93*, August 1993.

[19] Domenico Ferrari. Real-time communication in packet switching wide-area networks. Technical Report TR-89-022, International Computer Science Institute, Berkeley, California, May 1989.

[20] Domenico Ferrari. Real-time communication in an internetwork. *Journal of High Speed Networks*, 1(1):79–103, 1992.

[21] Domenico Ferrari, Anindo Banerjea, and Hui Zhang. Network support for multimedia: a discussion of the Tenet approach. Technical Report TR-92-072, International Computer Science Institute, Berkeley, California, October 1992. Also to appear in *Computer Networks and ISDN Systems.*

[22] Domenico Ferrari and Dinesh Verma. A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communications*, 8(3):368–379, April 1990.

[23] Alexander G. Fraser and Paul S. Henry. Transmission facilities for computer communications. *ACM Computer Communication Review*, 22(5):53–61, October 1992.

[24] Alexander G. Fraser, Chuck R. Kalmanek, A.E. Kaplan, William T. Marshall, and R.C. Restrick. Xunet2: A nationwide testbed in high-speed networking. In *Proceedings of INFOCOM'92*, Firenze, Italy, May 1992.

[25] P. Gautier and P. Chemouil. A system for testing adaptive traffic routing in france. In *Proceeding of GLOBECOMM '87*, Tokoyo, Japan, 1987.

[26] M. Ghanbari. Two-layer coding of video signals for VBR networks. *IEEE Journal on Selected Areas in Communications*, 7(5):771–781, June 1989.

[27] Michael Gilge and Riccardo Gusella. Motion video coding for packet switching networks – an integrated approach. In *SPIE Visual Communications and Image Processing '91*, November 1991.

[28] P. Humblet and S. Soloway. Algorithms for data communication networks - part i, ii. Technical report, Codex Corporation, Cambridge, Mass, 1986.

[29] G. Karlsson and M. Vetterli. Subband coding of video for packet networks. *Optical Engineering*, 27(7):574–586, July 1988.

[30] Mark J. Karol, Michael G. Hluchyj, and Sam P. Mogan. Input versus output queueing on a space-division packet switch. *IEEE Transactions on Communications*, 35(12):1347–1356, December 1987.

[31] V. Kompella, J. Pasquale, and G. Polyzos. Two techniques for multicasting for multimedia applications. In *Proc. of Third International Workshop on Network and Operating System Support for Digital Audio and Video*, San Diego, California, November 1992.

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

[33] A. Lazar and G. Pacifici. Control of resources in broadband networks with quality of service guarantees. *IEEE Communications Magazine*, 1991.

[34] K. Mase and H. Yamamoto. Advanced network traffic control methods. *IEEE Communications Magazine*, October 1990.

[35] D. Mitra, R. Gibbens, and B. Huang. Analysis and optimal design of aggregated-least-busy-alternate routing on symmetric loss networks with trunk reservation. In *International TeleTraffic Congress ITC-13*, 1991.

[36] Arun Netravali and Barry G. Haskell. *Digitial Pictures - Representation and Compression*. Plenum Press, NY, 1988.

[37] C. Parris, S. Keshav, and D. Ferrari. A framework for the study of pricing in integrated networks. Technical Report TR-92-016, International Computer Science Institute, Berkeley, California, March 1992.

[38] Colin Parris and Anindo Banerjea. An investigation into fault recovery in guaranteed performance service connections. In *Proceedings of ICC/SUPERCOMM '94*, New Orleans, LA, May 1994.

[39] Colin Parris, Giorgio Ventre, and Hui Zhang. Graceful adaptation of guaranteed performance service connections. In *Proceedings of IEEE GLOBECOM'93*, Houston, TX, November 1993.

[40] G. Parulkar and J. Turner. Towards a framework for high-speed communication in a heterogeneous network environment. *IEEE Network Magazine*, March 1990.

[41] R.F. Rey. *Engineering and Operations in the Bell System, Second Edition.* AT&T Bell Laboratories, 1983.

[42] Marshall T. Rose. *The Open Book: A Practical Perspective on OSI.* Prentice-Hall, Englewood Cliffs, NJ, 1990.

[43] Marshall T. Rose. *The Simple Book: An Introduction to Internet Management,2nd Edition.* Prentice-Hall, Englewood Cliffs, NJ, 1994.

[44] Samuel Sheng, Anantha Chandrakasan, and Robert W. Brodersen. A portable multimedia terminal. *IEEE Communications Magazine*, pages 64–75, December 1992.

[45] Atsushi Shionozaki and Mario Tokoro. Control handling of real-time communication protocols. In *Proceeding of the SIGCOMM '93*, September 1993.

[46] R. Stacey and D. SongHurst. Dynamic alternate routing in the british telecom trunk network. In *Proceedings of the ISS '87*, Phoenix , Arizona, 1987.

[47] William Stallings. *SNMP, SNMPv2, and CMIP. The Practical Guide to Network-Management Standards.* Addison-Wesley Publishing Company, Inc., 1993.

[48] Michael Stonebraker. An overview of the Sequoia 2000 project. In *Proceedings of COMPCOM 92*, San Francisco, CA, February 1992.

[49] F. Teraoka, Y. Yokote, and M. Tokoro. A network architecture providing host migration transparency. In *Proceedings of the SIGCOMM'92*, pages 209–220, 1991.

[50] Y. Tobe, H. Tokuda, S.T.C. Chou, and J.M.F. Moura. QoS control in ARTS/FDDI continuous media communications. In *Proceeding of the SIGCOMM '92*, pages 88–98, August 1992.

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

[52] Dinesh Verma, Hui Zhang, and Domenico Ferrari. Guaranteeing delay jitter bounds in packet switching networks. In *Proceedings of Tricomm'91*, pages 35–46, Chapel Hill, North Carolina, April 1991.

[53] Bernd Wolfinger and Mark Moran. A continuous media data transport service and protocol for real-time communication in high speed networks. In *Proceedings of Second Int'l. Workshop on Network and Operating Systems Support for Digital Audio and Video*, Heidelberg, Germany, November 1991.

[54] Nanying Yin and Michael G. Hluchyi. A dynamic rate control mechanism for integrated networks. In *Proceedings of INFOCOM'91*, 1991.

[55] Hui Zhang. *Service Disciplines for Integrated Services Packet-Switching Networks*. PhD dissertation, University of California at Berkeley, November 1993.

[56] Hui Zhang and Domenico Ferrari. Rate-controlled static priority queueing. In *Proceedings of IEEE INFOCOM'93*, pages 227–236, San Francisco, California, April 1993.

[57] Hui Zhang and Srinivasan Keshav. Comparison of rate-based service disciplines. In *Proceedings of ACM SIGCOMM'91*, pages 113–122, Zurich, Switzerland, September 1991.

[58] Hui Zhang, Dinesh Verma, and Domenico Ferrari. Design and implementation of the real-time internet protocol. In *Proceedings of IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems*, Tuscon, Az, February 1992.

[59] Lixia Zhang. *A New Architecture for Packet Switched Network Protocols*. PhD dissertation, Massachusetts Institute of Technology, July 1989.

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