FLEXIBILITY IN THE INTERACTIONS BETWEEN HIGH-SPEED NETWORKS AND COMMUNICATIONS APPLICATIONS

by

Paul Eric Haskell

Memorandum No. UCB/ERL M93/83

2 December 1993

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

FLEXIBILITY IN THE INTERACTIONS BETWEEN HIGH-SPEED NETWORKS AND COMMUNICATIONS APPLICATIONS

by

Paul Eric Haskell

Memorandum No. UCB/ERL M93/83

2 December 1993

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

Flexibility in the Interactions Between High-Speed Networks and Communications Applications

by

Paul Eric Haskell

Doctor of Philosophy in Engineering-Electrical Engineering and Computer Sciences

University of California at Berkeley

Professor David Messerschmitt, Chair

Recent research efforts in the design of both communications networks and applications have led to increased adaptability in both domains. Flexible networks support a large variety of applications efficiently and facilitate the introduction of new applications. Adaptable applications can use a wide variety of networks, such as wireless, local-area, and Broadband Integrated Services Digital Networks (BISDN's), without modification. Surprisingly little research has focused on the interface between applications and networks, however. Currently proposed interface models often are poorly defined or so simple as to hinder high application performance and efficient network resource use.

This thesis shows the feasibility and benefits of a richer channel setup interface by presenting a new interface model and then showing how video applications and networks could use the model to provide high-performance service with high transport resource uti-

lization. First, we propose the Medley Interface model, which combines *substreams* of several different transport qualities of service (QOS) into a single channel. The Medley Interface also proposes a detailed QOS description format that bounds each individual substreams' data rates, delays, and loss rates, and further allows bounds to be placed on the burstiness or spacing of substreams' losses. Loss burstiness control is beneficial to applications such as video or file-transfer whose performance varies as much with their channels' loss spacing as with their loss rates.

Next, the thesis presents a channel parameter negotiation method that reduces network resource requirements while maintaining a constant level of application performance. This iterative minimization technique achieves channel cost reductions ranging from 20% up to 70% with several applications; these negotiations require the detailed transport description provided by substream decomposition and the Medley Interface QOS format.

We present new variations of existing video coding algorithms that maintain good video quality over the range of channel parameters that might result from negotiations. *Leaky motion compensation* causes transmission errors to disappear quickly and smoothly. When performed adaptively based upon the coded scene's contents, the bit-rate penalty of leaky compensation can be made very small.

Finally the thesis presents several new buffer access disciplines that allow networks to provide channels with highly correlated or widely separated losses. These special-purpose disciplines allow networks to allocate as much as 50% less buffer space as would be needed with generic buffer disciplines.

Together, the network interface, video coding methods, and buffer management disciplines presented in this thesis show the benefits and feasibility of a richer call setup network interface than has been envisioned. Video applications can operate with a range of channel QOS parameters, but they must have some control of the parameters to adapt to produce video with high subjective quality after transmission. Networks can provide channels with delay, loss rate, loss priority, and loss spacing characteristics finely tuned to the needs of

specific applications, but these needs must be made known to the network. These channel characteristics are specified via the Medley Interface's substream decomposition and new QOS format; further, these components facilitate channel setup negotiations that minimize an application's transmission cost at a fixed performance level.

Approved:

David G. Messerschmitt 10/26/93

Chair Date

TABLE OF CONTENTS

LIST OF FIGURES

List of Tables

List of Image Photographs

Acknowledgments

I would like to thank the members of my thesis committee, and especially its chair Dave Messerschmitt, for their assistance and guidance during the completion of this project. I gratefully acknowledge the help and the company of my fellow students during my years at Cal. I especially would like to recognize John Barry, Joe Buck, Shih-Fu Chang, Wan-teh Chang, Rick Han, Alan Kamas, Allen Lao, William Li, Horng-dar Lin, Valerie Taylor, and Louis Yun.

I dedicate this work to my family and to Angela, in hopes that we will be family soon ♥

Chapter 1

INTRODUCTION

Traditionally, communications networks have supported only a single type of application. For example, the telephone network is designed to support voice communications; fairly elaborate processing is necessary to use this network to transmit computer data even at moderate rates. Computer networks can support high-bandwidth communications, but these networks are ill-suited to transport video and audio data because of their poor ability to control loss and delay characteristics. The specialized capabilities of current networks have led to the deployment of many parallel networks within the same area—computer networks, telephone networks, cable television networks, and safety and security monitoring networks all may exist within a single building. This duplication of transport capability is inefficient, and it hinders the introduction of new communications applications because each new application, collaborative multimedia for example, must find a new network with which to operate.

The Broadband Integrated Services Digital Network (BISDN) intends to eliminate the waste inherent in the provision and maintenance of numerous parallel single-use networks. BISDN's can transport data for applications with a wide range of transmission requirements; video applications that transmit tens of megabits per second and sensor monitors that sporadically transmit only a few bits can share the same BISDN fairly efficiently. BISDN's merge features of circuit-switched and packet-switched networks to gain some of the benefits of each. They transport data in the form of small, fixed-size bundles called *cells*, each of which contains a small header with network information that is used for routing, identification, etc. Since cells are of a fixed size, network switch and buffer architectures are not too complicated. Since sources can transmit cells at variable rates, network resources can be shared efficiently.

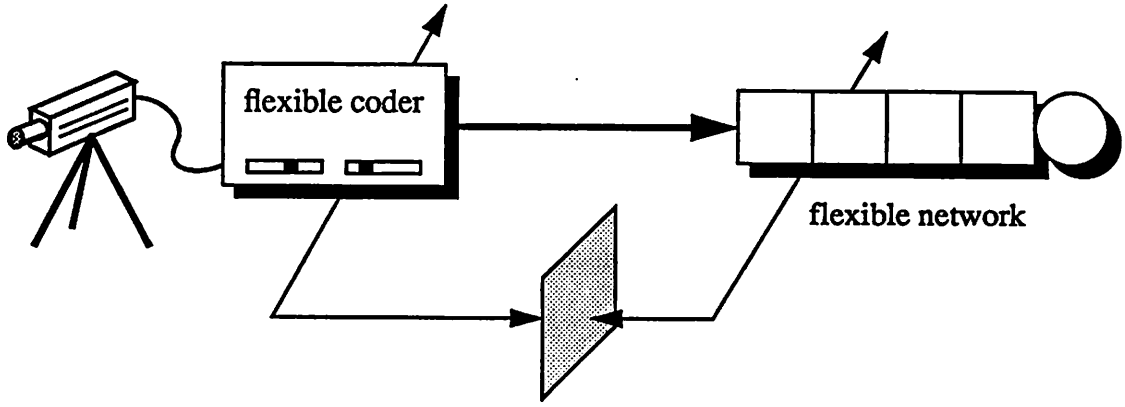
An application that uses a BISDN must tell the network about its data rate characteristics and its transmission quality requirements, and the network provides a channel that meets these needs. To do so a network must contain significant amounts of control intelligence. To avoid congestion, the network must route channels so as to balance the load through its switches and transmission links. The network must allocate sufficient resources to channels to ensure that the channels' transmission qualities are maintained but must not over-allocate resources, which would limit the number of channels that the network can provide. Also, the network must control the behavior of switch buffers and interconnect hardware so that buffers can be shared among channels even at high data rates without excessively degrading the channels' delays and loss rates. The control of networks to achieve these objectives is an active research area, and many significant achievements have been obtained in the past few years [33, 35, 39, 46, 49, 53, 54, 55, 56, 59, 63, 64, 72, 76, 77, 84, 85]. Useful reviews of this work are found in [38, 78].

Concurrently, many researchers have been studying how to make communications applications themselves more flexible and adaptable for a wide variety of needs. For example, two recently finalized video compression standards produced by the Joint Photographic Experts' Group (JPEG) and Motion Picture Experts' Group (MPEG) both specify several different compression modes, each of which is best suited for particular video source material, picture resolutions, or coded video quality [99, 103]. In fact, the recent high definition television (HDTV) format proposals presented to the U. S. Federal Communications Commission also adapt with respect to the format of the input material and the noise level of the through-the-air transmission channel available [92, 94, 96, 98].

Although increasing adaptability and flexibility are found both in modern communications applications and the networks they use, the utilization of this flexibility is constrained by the interface between applications and networks. Even in current BISDN proposals, applications must describe their rate characteristics and transport quality of service (QOS) needs in terms of a small, fixed number of parameters. Current interface proposals often are

inadequately defined, which prevents applications from relying upon the QOS specifications they receive from their networks. Other interfaces are so simple that they severely limit applications' and networks' ability to adapt to each other's characteristics. The restricted ability of an application to learn about a network's capabilities limits how much it can modify its signal processing and encoding to adapt to channel impairments. The limited ability of a network to learn about specialized needs of its clients limits its ability to meet those needs efficiently.

This thesis presents three methods that cooperate to improve efficiency in networks and their client applications: a flexible channel setup interface, video coding techniques that operate with a range of channel characteristics, and network buffer management disciplines that implement channels with a range of application-specific characteristics.



Three Methods for Improving Network and Application Efficiency (Fig. 1)

After a review of BISDN terminology and components in chapter 2, chapter 3 presents a flexible interface model, called the *Medley Interface*, for channel setup between communications applications and high-speed digital networks. The model fosters more elaborate communications between these two entities than suggested previously. The first part of the Medley Interface is a paradigm for the division of a channel into smaller units that can be described more easily and exactly than the channel as a whole. These pieces, called *substreams*, each have their own rate and QOS specifications. An application can request as

many substreams as it desires, and the application's channel properties follow from the properties of its component substreams. Substream decomposition allows networks to tune a channel's resource allocation and control to meet the specific QOS needs of an application's different data types. Without substreams, a network must give an application a channel tuned to the worst-case resource needs of all of the application's data types—if these needs vary widely, the channel will use resources very inefficiently. Further, substream decomposition decouples channels' descriptions from their network implementations, simplifying the design of applications for use with a variety of networks.

A channel's transport behavior can be divided into data rate, delay, and loss characteristics; the specification of these characteristics is called a channel's *flow specification* or *flowspec* [80, 86]. The second part of the Medley Interface model is a flowspec format that supports detailed specification of these channel characteristics. To date, several researchers have studied how to best describe data rate characteristics [44, 54, 55, 83], and the Medley Interface flowspec format uses these works. This thesis argues that the simple delay descriptions in use today are likely to be adequate for the near future. Most networking studies describe a channel's loss characteristics through its average loss rate, but the Medley Interface flowspec format describes a channel's loss characteristics in more detail. We have found several types of communications applications whose performances depend strongly on loss characteristics beyond the average loss rate. The Medley Interface flowspec format allows these types of characteristics, basically the spacing between cell losses and groups of losses, to be specified. Networks have the capability to implement this type of loss behavior, applications benefit from it, and with this description format applications and networks can cooperate to take advantage of these capabilities.

Different networks have different topologies, transport links, and switching resources. Thus, two different networks might implement two channels with identical flowspecs differently. If an application knew about its network's relative availability of transport link, buffer, and processing resources, the application might request a channel with different

flowspec parameters in order to best balance the trade-off between channel cost and the performance level that an application delivers to its user. Currently, it would be very difficult for an application to explore this trade-off since the only information it obtains about a network's resource availability is through the costs of channels with different descriptions.

The final part of the Medley Interface channel setup model is a flowspec parameter negotiation method that finds systematically the minimum cost channel for a fixed level of application performance. This iterative negotiation method requires that an application know how its performance varies as a function of its flowspec parameters. This knowledge can be embodied either in a function that assigns numerical performance values to sets of flowspec parameters or in sets of parameters that yield a constant performance level. The second approach is especially suitable for applications whose performances are evaluated subjectively, such as video or audio. Substream decomposition and the detailed flowspec format cooperate to enable negotiations to reduce channel resource requirements more than would otherwise be possible. The ability to describe application needs and network capabilities in detail allows negotiations to trade off among a wide range of channel characteristics.

The negotiation method does not require detailed information from a network as to its resource availability. The network simply must be able to tell an application the cost and *cost gradient* of a channel with a given description. A channel's cost gradient tells how sensitive its cost is to changes in flowspec parameters. During channel setup negotiations a network either can return cost and cost gradient information repeatedly as a channel's parameter space is explored, or the network can transmit a description of its cost function to a negotiating entity once at the start of call setup. The negotiation procedure uses a variant of the gradient descent minimization algorithm; at every iteration step the current flowspec parameters are refined such that the channel cost decreases but the application performance remains constant.

We have implemented channel setup negotiations for several types of communications

applications and for networks with varying cost functions. For these examples the negotiating process does succeed in reducing channel costs while maintaining a near-constant application performance level.

To justify the performance gains possible with this flexible network interface model, this thesis must show how real-world applications can adapt to a variety of networks. Video applications are used as examples frequently throughout this paper—these are among the most demanding applications that will be carried by future high-speed digital networks because of their high bandwidths, tight delay requirements, and relatively stringent loss requirements. Multimedia applications perhaps make an even stronger argument for flexible network interfaces, however. Multimedia applications integrate the exchange of multiple data types such as video, still images, audio, text, graphics, and control; each of these data types has unique rate and QOS needs. A network that can describe these needs exactly and that can transport these data types over connections that are tuned to their needs can operate much more efficiently than a network that implements a single monolithic connection that must meet the most stringent requirements of all of the data types.

In the spirit of the flexibility shown by the MPEG and JPEG video coding standards, chapter 4 presents some modifications of commonly-used video compression techniques that allow video coders to maintain a constant perceived quality when using transmission channels with varying loss characteristics. These techniques are used as part of a Medley Interface channel setup negotiation to show how a flexible video coder can operate with a variety of networks.

In addition to not allocating resources for worst-case QOS needs, a network can reduce a channel's resource allocation by employing buffer management disciplines that tailor the channel's loss and delay characteristics to specific application needs. There have been a number of research efforts that present buffer management disciplines that are optimal in some sense [35, 64, 75, 85]. However, these works have not proposed that different disciplines may be appropriate for different applications. Chapter 5 presents some new special-

ized buffer management disciplines that implement channels with the specific types of loss characteristics described by the Medley Interface flowspec format. With a flexible network interface, applications can request channels with different types of loss control for different substreams. Networks can implement these requests by using different buffer management disciplines for different substreams, including the new buffer management disciplines presented here. For example, a network could use a buffer management discipline that prevents consecutive cell losses to reduce the buffer requirements of a high-performance video application.

The flexible video compression methods and buffer management disciplines presented here are useful in their own right, but they also serve to show the advantages of a more powerful channel setup interface than previously has been proposed. A more general interface allows applications to specify their transport QOS needs more exactly, and enables more efficient resource allocation and greater resource savings during flowspec parameter negotiations. Networks can use special-purpose methods for implementing channels with the negotiated characteristics, and applications can adapt to the specified channel properties. Perhaps more importantly, as new applications are developed and as new network capabilities are discovered, they can be integrated into an existing infrastructure without major disruption.

Chapter 2

CELL RELAY NETWORKS

The introduction stated that modern digital networks increasingly are able to adapt their behavior to suit the needs of a wide variety of communications applications. This section reviews some properties and components of these networks and discusses the control techniques that allow networks to guarantee the quality of service (QOS) that they offer their clients. This chapter also gives an overview of some methods commonly used by networks and applications to alleviate the effects of data loss on the application's delivered performance. Finally, this chapter presents some recent research in the design of the network-application interface. Later sections of this report present a new interface model that allows networks and communications applications to take advantage of the flexibility and adaptability they increasingly possess.

2.1 Asynchronous Transfer Mode

Many types of electronic data networks, such as telephone and broadcast television networks, are *circuit-switched.* For each connection, the network allocates a fixed "circuit" consisting of bandwidth and switching resources, connection identifiers, etc. Networks can decide whether or not to accept a new connection easily, since each connection consumes a fixed amount of resources and since networks know how much of their resources are unused. Signals that contains time-varying amounts of information must be smoothed with buffering or variable-quality coding so that they can be sent over the fixed bandwidth given by a circuit-switched channel.

Computer data networks transport very bursty traffic. Keystrokes from a human user, graphics data from a drawing program, and electronic mail are all examples of traffic sources in which the data rate for some short time intervals is much higher than for others. To

handle bursty sources efficiently, computer networks employ *packet-switching*, in which data streams are divided into bundles called *packets*, each of which contains a header with routing information, priority identifiers, etc. Since each packet can be transmitted as a stand-alone unit, network resources such as transmission lines and buffers need not be reserved for specific connections; resources can be shared among several connections efficiently.

Many computer networks compute each packet's route independently. Routing is computationally expensive, but at low to moderate speeds networks are able to route once per packet.

Some networks use variable-sized packets. This allows small bursts of data to be sent in a small packet—they need not wait for more data to arrive to fill a larger packet. Also, large bursts of data can be sent in a single packet, to minimize header space and routing computations. However, the design of network switches and buffers is complicated with variable-length packets.

Cell-relay networks merge features of circuit-switched and packet-switched networks to gain some of the benefits of each. Cell-relay networks divide traffic from each source into small, fixed-size bundles called *cells*, each of which contains a small header with network information. Since the cells are small, sources that generate data sporadically need not wait too long before filling them. Since the cells are fixed-size, the designs for network switches and buffers are simpler than for packet-switched networks.

The Broadband Integrated Services Digital Network (BISDN) combines cell-relay transport with *asynchronous transfer mode* (ATM) switching to achieve high network resource utilization and thus great efficiency [61, 67]. ATM switching allows sources to transmit cells not periodically but whenever data are available. For example, an ATM speech coder would not output cells when the speaker is not talking. With circuit-switched networks sources often have to transmit "dummy data" to satisfy those networks' constant bit-rate requirement.

The ability to transmit variable-rate sources gives ATM networks a *statistical multiplexing* gain. An ATM network allocates each source some bit-rate below its peak rate but higher than its average rate. It is statistically unlikely that all sources simultaneously transmit at their peak rate, so almost always the aggregate rate of all of the network's sources is less than the network's capacity. The network thus can carry more channels than if each source were allocated its peak bit-rate. When many channels are sent through the same network nodes and buffers, statistical multiplexing allows two or three times as many channels as if peak-rate allocation were used [21, 76, 89].

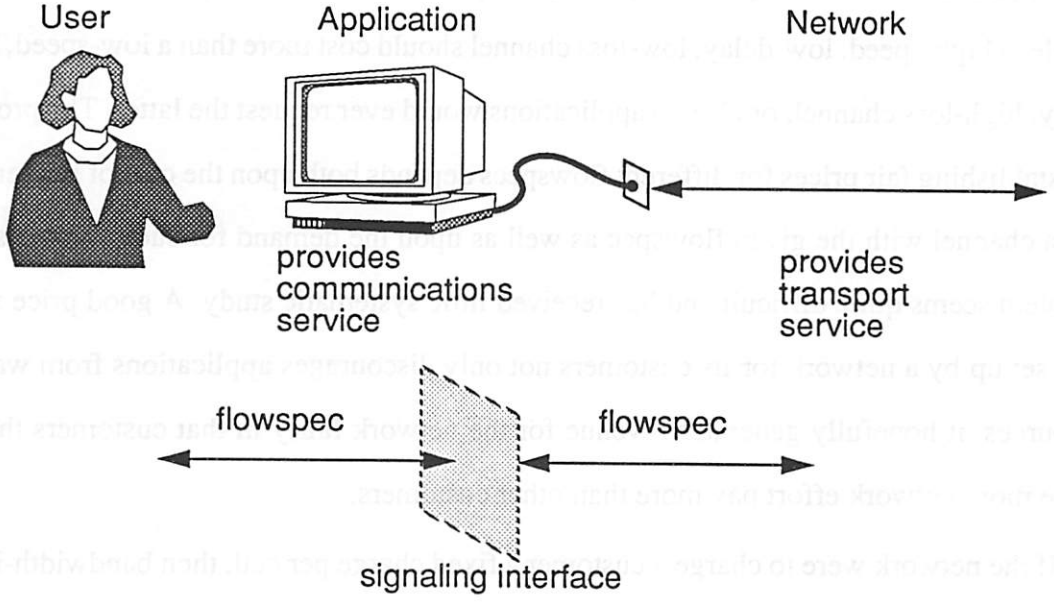
ATM networks, similarly to circuit-switched networks, establish connections called *virtual circuits* that last for the duration of an application's communications rather than for a single cell time. Routing is performed once per virtual circuit rather than once per cell. During channel establishment, all network nodes that implement a virtual circuit perform routing (e.g. with the shortest-path algorithm [38]) and associate the virtual circuit identifier with the correct routing path. During cell transport, nodes use a table to translate virtual circuit identifiers to output links; no routing algorithm need be performed.

Also during channel setup, network nodes may allocate buffer space or other resources to a virtual circuit. In fact, during virtual circuit establishment, the source and network may negotiate average and peak source cell rates, allowed network delay, loss rates, etc. Many applications need these quality of service (QOS) guarantees from the network in order to function usefully. For example, since people find two-way voice communications quite awkward if the end-to-end delay in the network exceeds 0.5 seconds, a voice coding application should negotiate with the network to obtain a channel with a smaller delay.

2.2 Network Components

We define an *application* to be hardware or software that provides some communications service to a human *user*. Examples of applications include multimedia editors, videoconferencing systems, remote visualization devices, file-transfer software, etc. An application's *performance* is a measure of how well it satisfies its user—performance could

be the subjective quality of an audio or video presentation, the throughput rate of a file-transfer application, etc. A network provides *transport services* to its client applications. Transport services include sequenced cell delivery, delivery at specified loss rates and delay bounds, delivery with lost cell notification, etc. A network implements a *channel* to provide a transport service to an application. A channel's *QOS specification* or *QOS guarantee* defines worst-case loss and delay characteristics that a network promises to give the channel; the channel's *QOS* is the characteristics it actually receives. A channel's QOS specification combined with the data rate bounds that its client application promises to observe constitute the channel's *flow specification* or *flowspec* [80, 86]. A channel's flowspec is established over a *signaling interface* during channel establishment.



Terminology (Fig. 2)

A cell-relay network consists of end-user equipment, network switches, links that connect the user equipment and switches, and management entities that control the network. The links and switches have finite capacities, measured in bits per second or cells per second. One of the hardest problems in the field of data networks is to decide how to choose an interconnection topology among the various network components and link bandwidths and switching resources within the topology so that the network can handle a maximal

amount of application traffic. This question is beyond the scope of this report, but [38, 63, 77] provide an introductory discussion.

Management functions within the network are responsible for ensuring that the network runs properly. These operations include fault detection, billing, and QOS monitoring.

2.3 Channel Setup

During channel setup, an application and the network agree on a flowspec, and the network tries to establish a suitable channel. Flowspec parameters include network QOS measures such as fixed or probabilistic bounds on end-to-end delay, variability in delay (called *delay jitter*) and probability of cell loss. The network must establish a pricing structure so that applications do not request a higher QOS specification than they really need. For example, a high-speed, low-delay, low-loss channel should cost more than a low-speed, high-delay, high-loss channel, or else no applications would ever request the latter. The problem of establishing fair prices for different flowspecs depends both upon the cost of implementing a channel with the given flowspec as well as upon the demand for such channels; this problem seems quite difficult and has received little systematic study. A good price structure set up by a network for its customers not only discourages applications from wasting resources, it hopefully generates revenue for the network fairly in that customers that require more network effort pay more than other customers.

If the network were to charge a customer a fixed charge per cell, then bandwidth-intensive applications such as video would subsidize less bandwidth-intensive applications. Call set-up, network maintenance, and network overhead consume network resources; pricing structures should reflect these expenses. Also, true per-cell charging is probably too difficult for networks to implement.

If prices were proportional only to call duration, then applications would have no incentive not to transmit data at very high rates, wasting network resources. The easiest way to base pricing is on the agreed-upon flowspec parameters as well as on the channel lifetime. Of course, if an application violates its rate parameters, the network could impose sur-

charges instead of simply discarding the excess traffic. If the network violates its QOS guarantees, then the client could receive a partial refund of charges.

2.3.1 Rate Description and Quality of Service

During call setup, an application tells its network its traffic rate characteristics and requests a certain channel QOS specification. The network uses this information to establish a suitable channel. It would be useful if a network could deduce an application's traffic rate characteristics directly from its transmitted data stream. However, the network must know the application's traffic description before it creates a channel.

An application could specify only its peak data rate in describing its traffic. Then, the network pessimistically must assume that the application always transmits at this peak rate. It is more common for broadband network proposals to require that applications specify both their peak and average data rates [104]. With these two metrics, networks can take advantage of applications' time-varying resource needs.

The *leaky bucket* is a common method for traffic description [46]. The leaky bucket acts like a buffer with a fixed maximum size and deterministic service rate. This monitor requires that a source's cell traffic not overflow a buffer with a specified capacity and service rate.

To implement a peak rate constraint, a leaky bucket monitor with a capacity of one and a service rate equal to the peak rate can be used. If two cells arrive spaced apart by a time less than (1 / *peak rate*) then the leaky bucket length would grow to 2 and the capacity would be exceeded. To implement an average rate constraint, a leaky bucket with a large capacity and rate equal to the average source rate can be used. Even if the source has long bursts which exceed the average rate, the bursts should get absorbed in the large leaky bucket capacity. If the bucket ever does overflow, then the source has exceeded its average rate for a long time.

Multiple leaky buckets would be helpful in describing the traffic from a single source.

The peak and average rate of a source could be monitored with the combination of the two leaky bucket monitors discussed above. Other two-bucket rate descriptions could be more useful to the network, however. Possibly, rate descriptions with more than two leaky buckets could be employed to allow the network to obtain detailed information on the traffic rate statistics of a particular application. For example, a network could ask an application for appropriate leaky bucket rates for bucket sizes equal to the buffer sizes of each network switch in a given channel.

Communications applications must know how to translate their high-level performance requirements into network flowspec' parameters. For example, a video coder must translate performance requirements such as interactive response time and image fidelity into flowspec parameters such as channel delay and cell loss probability. The more accurately that an application can describe its QOS needs to the network, the less the application must overestimate its needs, incurring extra cost and wasting resources.

2.3.2 Resource Allocation and Routing

While establishing a connection to a particular destination, a network performs *routing* by choosing a path of switches and transmission links that connects the source to its destination. Concurrently, the network must ensure that adequate resources exist along the route to carry the source's data. If no route exists with adequate resources, the source must be informed that its request for a connection must be denied. Optimally, the network combines knowledge about its own topology, its available resources, and the likely pattern of future channel requests to choose routes that will not prevent future channels from being established.

The network determines how much buffer, switching, and bandwidth resources to allocate to a channel based on the channel's rate description and QOS specification parameters. If the network allocates resources aggressively, assuming that the channel will not need many resources, then the network can accept more channels than a conservative network. However, applications' QOS guarantees will be violated more often by an aggressive net-

work than a conservative one.

It would be possible for networks to allocate resources to channels dynamically as they are needed. However, dynamic resource allocation creates the problem of what to do if needed resources are not available. Most users would prefer to be denied a connection at the start of channel establishment rather than to have their channel break down in the middle of use.

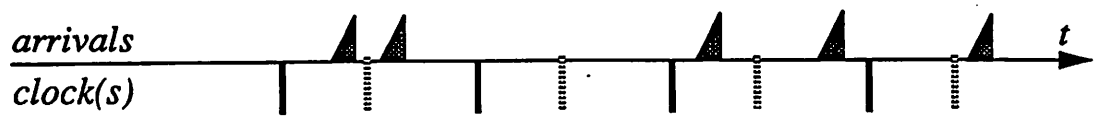
2.4 Rate Monitoring

The resources allocated by a network for a specific channel only will be adequate to guarantee a certain quality of service if the source traffic conforms to the rate description that it promised. To ensure that sources are well-behaved, networks use a *rate monitor* or *policing agent.*

The leaky bucket described in section 2.3.1 commonly is used as a rate monitor. Again, the leaky bucket acts as a finite sized buffer with a fixed service rate. To do rate monitoring, the leaky bucket need not actually buffer any cells. The leaky bucket can be implemented with an up-down counter that is incremented whenever the source being monitored outputs a cell and that is decremented periodically at the service rate. If the counter counts down to zero, it is not decremented further. If the counter increases to the leaky bucket size, then any cells that arrive when the bucket is full should be discarded rather than given to the network. The count of the leaky bucket at all times is equal to the fullness of an actual queue with the same service rate and size as the leaky bucket.

The leaky bucket is a simple and effective policing agent. However, one drawback is that with large bucket sizes (as would be used to verify an average-rate bound), the network cannot detect a violation until the violation has continued for a significant time. Also, the relative time-phase of the leaky bucket decrements with respect to the source traffic affects

the monitor's decisions.



Different Leaky Bucket Phases (Fig. 3)

In figure 3, the triangles mark cell arrival times. The solid and dashed lines indicate decrement times for two leaky buckets. Both leaky bucket monitors operate at the same service rate. However, if both leaky buckets have size 1 (i.e. if they monitor a peak rate constraint) then the leaky buckets make different decisions. The leaky bucket corresponding to the solid lines rejects the second and fourth arrivals. The other accepts all of the arrivals. Because of this timing phase sensitivity, it is common with the leaky bucket as well as with other policing methods to make the policing agent parameters somewhat less stringent than the negotiated rate description parameters. When allocating resources for new channels, a network must know about this safety margin built into its policing agents.

Next, we review policing methods other than the leaky bucket [72, 79]. With the *jumping window* method, the number of cells that a source can transmit in fixed-size time windows is limited to a maximum value. If more cells arrive in a window than is permitted, the extra cells are discarded or marked for possible future discard. Each time window starts immediately after the preceding window. With this method, the time windows are not synchronized with cell arrivals at all. The *triggered jumping window* policing method differs from the jumping window method in that a window interval does not start until the first cell arrival following the end of the previous window. The *moving window* method stores the arrival times of the previous N cells at all times, where N is the window size. If, at any cell arrival time, the most recent N cells have arrived in less than the prescribed window width, then the current cell is discarded or marked. Since this method requires the storage of possibly numerous cell arrival times, it is more complicated to implement than the previous policing methods.

The *exponentially weighted moving average* policing method is quite similar to the

jumping window method except that the number of cell arrivals in any window interval depends upon the number of arrivals in previous intervals as well. The more cells that arrived in previous intervals, the fewer cells are permitted to arrive in the current interval.

All of these policing methods can monitor peak cell arrival rate violations effectively; in fact the methods are nearly equivalent with a window size of 1. However, the leaky bucket method usually detects violations of mean arrival rate agreements more quickly and/or more accurately than the other methods [72].

Applications can use knowledge of a network's rate policing method to improve their performance. For example, video coders that use fixed-rate channels maximize their performance by always broadcasting at the maximum possible rate. With leaky-bucket policing however, a coder can "loan" itself bandwidth in the future by broadcasting slower than the bucket service rate; the bucket fullness then decreases to zero. When a scene change or burst of activity occurs, the video coder can use this "loaned" bandwidth to improve its performance. Note that since policing occurs at the source, the source should be able to prevent any discard by the policing agent by implementing its own copy of the policing agent and reducing its data rate when its copy is close to its limits. This method is similar to the use of buffer-length feedback in a fixed-rate channel system. However, broadband network rate monitoring may exert different influences on coders; if a network uses a leaky bucket controller to monitor a video source's average cell rate, the leaky bucket size likely would be larger than any physical buffer in a fixed-rate network.

2.5 Methods for Alleviating Data Losses

Some applications may transmit data of varying importance over a single channel. This section discusses some methods that networks and applications can implement to protect more important data from cell losses and also to make the effects of losses less severe.

An application could request two or more channels, each with different QOS specifications, for data streams with different loss sensitivities. However, the application might obtain some of the needed transmission channels but not others. Also, the application would

need to perform cell resequencing at the receiver since delays on different channels may differ. So, there are advantages to having applications establish only one transmission channel.

2.5.1 Priorities

Applications can use *priority notification* to specify that some transmitted cells are more important than others; networks then can offer different QOS guarantees to different priority traffic within the same channel. A simple scheme is proposed for the BISDN: one bit in each cell header specifies if a cell is "loss-eligible" or not. Within network nodes, if some number of cells from one source are to be discarded in order to prevent buffer overflow, the "loss-eligible" cells are discarded first. Thus, if a source marks all of its outgoing cells "loss-eligible" or "loss-ineligible", then cells are discarded at random. The fewer cells are marked "loss-ineligible", the less likely that loss-ineligible cells will be lost. However, the percentage of loss-ineligible cells discarded depends upon the state of the network as well. It is an unanswered question how high a percentage of loss-ineligible cells a coder should generate to protect them adequately—researchers have tried values from 0% past 50% [2, 16].

A video coder based on motion-compensation and the discrete cosine transform (DCT) could use priority notification to request that high-frequency DCT coefficients be discarded before motion vectors in the event of network congestion; the DCT coefficients have less influence than the motion vectors on the received video quality. Later in this report, we study how a more general multiple-channel network interface can allow modified video coders to send data over lossy networks more cheaply and with more loss-immunity than is possible with present networks. By sending data over channels with appropriate loss, delay, and bandwidth characteristics, video coders can transmit high-quality sequences efficiently in spite of cell loss.

A network policing agent need not discard cells in excess of the source's guaranteed rate; it could use the priority bit to mark such cells as loss-eligible. Then, if network re-

sources are free to deliver the cells even though they are in violation, they will be delivered. If the resources are not available, the illegal cells are discarded to prevent affecting other sources' QOS.

2.5.2 Error Correction and Detection

Forward error correction (FEC) techniques add coded bits to a data stream such that limited numbers of bit errors affecting either the new bits or the original input can be detected or corrected. Reed-Solomon coding, BCH codes, and Hamming codes all are examples of FEC techniques [107]. FEC increases the amount of data that a source must transmit and imposes coding delay also. Further, FEC codes that could correct for the loss of large numbers of bits, such as would occur with a cell loss, would be fairly complicated.

Nevertheless, some researchers have investigated FEC for recovery from cell losses [71]. Although this paper finds that FEC can reduce a channel's apparent cell loss rate, the paper does not address the bandwidth penalty that FEC imposes and does not mention how FEC could cope with consecutive cell losses.

Applications that cannot tolerate any losses but that can tolerate high delay, such as file-transfer, combine error-detection with bidirectional protocols that allow a receiver to request retransmission of corrupted data [38]. The overhead and expected delay of these *automatic repeat request* (ARQ) protocols can be traded against each other for efficient operation with a variety of networks.

2.5.3 Data Interleaving

Because cell losses often are caused by overflowing buffers within the network and because the conditions that cause buffer overflow may persist for a significant time, cell losses do tend to occur in bursts. If a source interleaves or shuffles its data before packing the data into cells, a burst of cell losses can be spread out. For applications such as video transmission this is helpful since video receivers can estimate lost data fairly well if the data in nearby picture regions is available. If a burst of cell losses were to corrupt the data for a

large picture region though, the coder could not reconstruct the data.

Data interleaving cannot reduce the number of cells lost on a channel, however. Further, interleaving imposes delay and storage penalties at both ends of a transmission channel. Finally, a communications application must know the statistics of cell loss burst lengths in order to decide the length of time over which to interleave its data.

2.5.4 Traffic Shaping

Either a data source or a network can spread out high-rate bursts from a source's cell stream by performing traffic shaping. A traffic shaper is a storage buffer at the beginning of a channel that absorbs rapid bursts of traffic and outputs them at a lower rate; the smoothed cell stream is less likely to cause network buffers to overflow than the original stream. Although traffic shaping probably makes sense for very bursty sources, this technique does add delay to a channel. Further, the memory required to perform traffic shaping might be better used to increase the size of network buffers. If the memory were used within network switch buffers, it could be shared among multiple applications and would only add to channel delay when switch buffers were close to overflowing.

2.6 Cell Switching

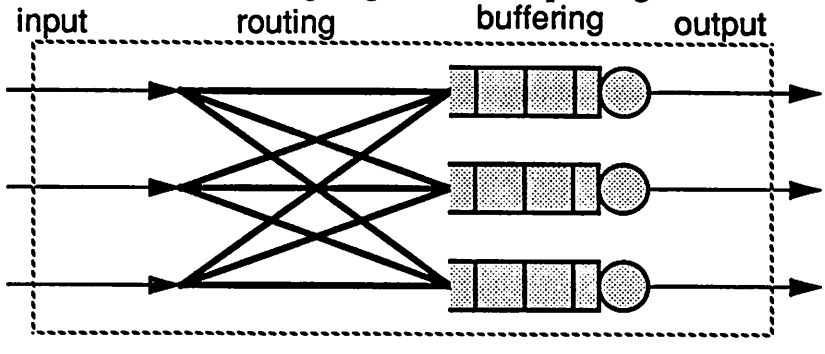
A network switch accepts cells from a number of input channels, determines where each cell should be sent next, and sends the cell to the appropriate output port some time later. From the output port, each cell travels over a transmission link to another switch or to its final destination. Much of the difficulty in switch design arises because for brief periods of time, the amount of input traffic that must be directed over a particular output link exceeds the link capacity. To alleviate this problem, cell-relay network switches contain buffer space in which cells can be stored until their output links become available. A switch's *buffer access discipline* decides whether an arriving cell will be stored in a buffer to await output or will be discarded because inadequate resources exist to process the cell. A switch's *buffer service discipline* determines when cells stored in one or more buffers

will be output.

During channel setup, a network uses the channel's QOS specification and rate description to allocate resources such as link bandwidth and buffer space within switches to the new connection. In conjunction with the rate monitor, a network can maintain the channel's QOS at each buffer and link with appropriate buffer access and service disciplines. For example, if a buffer fills so that it is in danger of overflowing, cells from different sources will be discarded in such a way that none of the sources' negotiated QOS guarantees are violated.

Switch architectures typically contain buffering components, routing logic, and control. Switches that first buffer all input cells before routing them to the appropriate output can suffer "head-of-line" blocking, when the cell at the head of a queue can not be output yet because its output port is busy and all cells from the same input are stuck waiting in the queue even though they are destined for different outputs. This is not efficient.

Numerous switch architectures buffer cells *after* routing them either partially or completely. This paper does not discuss switching architectures in detail (see [43, 57] for more information), but assumes a generic switch model that consists of an input stage, routing logic that operates at a high enough speed so that all input cells can be routed to the correct output in one cell duration, a buffering stage, and an output stage.

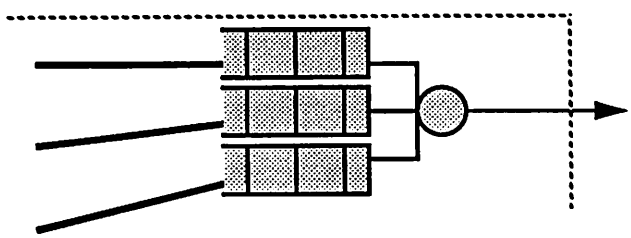


Switching Architecture (Fig. 4)

The above architecture shows a single buffer at each output port that is shared by all inputs that leave through the same port. Shared buffers gain a statistical multiplexing ad-

vantage; since it is unlikely that all or even most of the sources sharing a buffer simultaneously send data near their peak rate, then shared buffers can be smaller than dedicated ones.

However, it is difficult for a switch to guarantee equal quality of service to all input streams going to a common output if cells from all streams are stored in a common buffer. Switch service disciplines that can guarantee fair service to all inputs typically store cells from different inputs in different buffers (figure 5), and then specify the order in which the buffers are served. Since buffers are not shared by these disciplines, it is possible that an arriving cell is discarded because its input buffer is full even though space exists in other buffers. It should be noted that "fair service" is different from "least cell loss". A switch might lose the fewest cells by discarding inputs from only one source. This approach would not be fair to the penalized source however.



Multiple Buffers and A Server at One Output Port (Fig. 5)

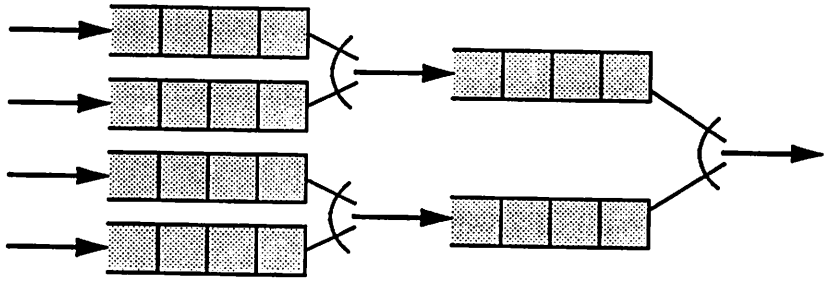
Buffer service disciplines that do not share buffers include the *virtual clock* and *fair queueing* methods, forms of the *earliest due date* strategy, the *stop and go* algorithm, and the *hierarchical round robin* strategy [85]. The virtual clock and fair queueing disciplines essentially are identical. These methods share bandwidth equally among all sources. However, if a source is not using its entire allotment, then other sources can share the unused bandwidth equally.

The "delay" version of the earliest due date strategy negotiates a transmission delay bound and transmission rate with each application. Arriving cells from each source are assigned a deadline which depends on the delay bound and agreed transmission rate. The server then sorts the arriving cells from all customers by deadline and transmits the cell

with the earliest deadline. The "jitter" version of the earliest due date strategy is similar to the "delay" version except that each switch can store cells in a buffer in order to provide minimum as well as maximum delay bounds.

The stop and go discipline divides time into contiguous frames. In each frame, only arrivals from the previous frame can be output. By adjusting the frame duration, this discipline can provide minimum and maximum delay bounds to its customers. Note that this method, as well as the jitter version of the earliest due date strategy, may leave output transmission links idle even when cells await transmission. Such service disciplines are called non-work-conserving.

The hierarchical round robin discipline is also non-work-conserving. This method maintains separate input queues for each customer. Each queue is served in turn, periodically. Several collections of round-robin queues can be served by a higher-speed round robin queue.



Hierarchical Round Robin Queues (Fig. 6)

Of course, buffer service disciplines can compromise between the benefits of shared buffers and buffers reserved for each input. For example, an output port could have one small buffer for each input and one additional large buffer for overflow cells. A possible modification of the round-robin service discipline for this arrangement would serve the overflow buffer whenever the dedicated input buffer to be served is empty.

The simplest buffer access discipline is *first-come first-served* (FCFS). An FCFS buffer serves cells in the order in which they arrive, and arrivals at a full buffer are discarded. Since FCFS never leaves buffer space empty when there are cells to be stored, this simple

discipline achieves the minimum possible cell loss rate.

A simple modification of the FCFS discipline that allows for cells of different priorities is called *FCFS with pushout* [64]. This discipline works much like simple FCFS, but when a cell arrives at a full queue it can push out a previously queued cell of lower priority if one exists. Most sensibly, the queued cell with lowest priority is dequeued and discarded. FCFS with pushout suffers an overall cell loss rate equal to that of ordinary FCFS. However, FCFS with pushout gives communications applications more control over which cells are lost.

The *partial buffer sharing* discipline [64] prevents lower-priority cells from entering a queue as the queue nears capacity. A partial buffer sharing queue of length L can be described with a *screening function* $r_L(l)$ that specifies the minimum priority level that a cell must have to gain admission to a queue of length L when that queue contains l cells.As l increases, cells need higher and higher priority levels to be admitted to the queue. $r_L(0)$ should equal the lowest priority in the system, and $r_L(L)$ should be higher than the highest priority in the system. For example, in a system in which priorities range from 0 to p_{MAX}, $r_L(l)$ could be the function $max(0, p_{MAX} - L + l)$ or $\frac{l \cdot p_{MAX}}{L}$.

This scheme differs from FCFS with pushout in that partial buffer sharing does not queue some low-priority arriving cells even when the queue is able to store them; the queue space is reserved for future higher-priority arrivals. Thus, partial buffer sharing results in more cell losses than FCFS with pushout. However, partial buffer sharing is simpler to implement than FCFS with pushout because the priorities of the queue contents need not be examined with partial buffer sharing. With both schemes however, when a queue is nearly full then only the highest-priority traffic is delivered, as desired.

2.7 Prior Interface Models

The goal of a network interface is to enable diverse types of applications to use a common network. Typically, existing interface models standardize a set of network transport service classes or network actions that applications can use for their communications needs.

The two interface models discussed next provide higher-level transport services than simple cell transport.

Proposed BISDN standards divide networking functions into separate protocol layers: the "Physical" layer transmits data units between connected network nodes and the "ATM Layer" transports cells between channel endpoints. The "ATM Adaptation Layer" (AAL) protocols work at a higher level than the ATM cell transport layer. AAL protocols add application-specific customization to the underlying ATM transport. For example, an AAL can perform segmentation and reassembly of application data units, extract timing information, and perform error-detection. Currently, four AAL protocols have been defined. AAL1 supports connection-based, constant-bit-rate communications applications. AAL1 supplies a timing signal, either synchronous or asynchronous to the network clock, to the receiver. Further, AAL1 segments and reassembles application data and maintains constant bit-rate delivery with buffering and bit-stuffing.

AAL2 supports connection-oriented applications that require timing information at the receiver but that do not require constant bit-rate delivery. The next ATM Adaptation Layer is called ATM3/4 for historical reasons. This layer supports connection-based or connectionless variable-rate data transfer. AAL3/4 performs data segmentation and reassembly, error-detection and possible retransmission of erroneous data, quality of service monitoring, etc. AAL5 also supports connection-oriented variable rate data transfer, but this AAL is simpler than AAL3/4. AAL5 does not support error recovery methods such as retransmission, but may notify the receiver of missing data. AAL5 also may offer applications less control over data transport than AAL3/4. For example, Bellcore proposes that applications that use AAL3/4 be allowed to cancel the transmission of in-progress frames, while applications that use AAL5 cannot [104].

The motivation for the AAL concept is that if a network can standardize several types of transport rather than just one, then the needs of more applications can be met more closely. This approach should be simple to implement because of the small number of proposed

transport service types, but the drawback is that the offered AAL transport service definitions do not meet the needs of complicated applications very well. AAL-based networks must over-allocate transport resources for applications whose quality of service needs cannot be met exactly.

Bellcore also was integral in the development of the "intelligent network" interface concept [36, 37, 51, 58, 74]. The intelligent network model defines a set of *functional components*, basic building blocks of telecommunications applications such as "join connections," "retrieve information from database *X*," "update record *Y*," "play audio announcement," "collect touchtone digits," etc. The standardization of functional components should facilitate the introduction of new applications, because part of the implementation of new applications can be defined in terms of already-implemented simple building blocks. In practice, new applications that require network actions not specified by standardized functional components would face a major obstacle to implementation.

Next we discuss several signaling interface models that support cell-relay channel establishment for a variety of services; none of these models completely solves the signaling interface design challenge though. Topolcic points out that no one yet has enough experience with large-scale cell-relay networks to specify fully how they should operate. "Few networks in the Internet currently offer reservation, and none that we know of offer reservation of all the resources specified here... No network [yet] provides for the reservation of packet switch processing bandwidth or buffer space." [80, p. 20].

The "Zeus" project at Washington University has proposed a flexible channel setup model for multiway connections. This project focuses on the multicast aspect of call control rather than on QOS specification [39, 41]. The Zeus project includes signaling protocols and switch hardware architectures that facilitate the establishment and maintenance of connections between numerous endpoints. Applications' only control their received QOS through the selection of one of four bandwidth choices, however.

Bellcore's "Expanse" project is similar to the Zeus project in many ways [66]. The Ex-

panse project defines an object-oriented call establishment model that implements complex, multiway call establishment. The model also supports higher-level application-specific transport services such as translation between different video formats and verification of end terminal capabilities. The project proposes a call setup language that is tuned to minimize the amount of call establishment traffic that must be exchanged between applications and networks.

Quality of service specification within the Expanse model is kept very simple. Applications choose from a small, fixed set of QOS specification options for each of their connections. Each connection between different endpoints can have its own unique QOS specification, however.

The Tenet group at U. C. Berkeley has proposed a network model and suite of protocols that implement channels with guaranteed QOS [48]. This work stresses the importance of QOS guarantees, implemented with a combination of call admission, rate monitoring, and resource reservation; several high-speed network test-beds demonstrate the feasibility of the Tenet channel model. The Tenet group's approach differs from that presented here in that they use probabilistic bounds on QOS measures such as delay or loss rate; probabilistic bounds are difficult to define in such a way that applications can rely on them to guarantee performance. Further, the Tenet group uses different flowspec parameters than those presented in chapter 3.

Two protocols for the Internet use concepts that should be useful in the BISDN also. The Experimental Internet Stream Transport Protocol, Version 2 (ST-II) proposes a multicast channel setup model that allows different participants in a communications session to use channels with different QOS specifications [80]. The protocol supports addition and deletion of participants in an ongoing multiway dialog and implements fault recovery also. ST-II supports guaranteed QOS channels as well as "best-effort" channels for which inadequate resources are reserved to meet their QOS guarantees.

The ST-II document presents a flowspec model but states that modifications are likely.

Flowspec parameters include cell loss probability, maximum channel delay, requested and minimum acceptable bandwidth, and "duty factor," the estimated percentage of time that the requested bandwidth actually will be in use.

The RSVP internet protocol presents a novel and clever multicast resource reservation protocol in which data receivers rather than sources set channel QOS requirements [86]. This protocol defines "filters" that allow different receivers in a multicast session to specify subsets of transmitted data that they wish to receive; these filters aid resource sharing in multicast connections. RSVP leaves the specification of flowspec parameters to its client applications, however.

2.8 Conclusion

ATM networks support a variety of communications applications, including video, effectively because they handle high-speed variable-rate traffic. Since ATM networks dynamically share resources among multiple sources, they use transmission and switching resources efficiently. Thus, ATM networks effectively can support multimedia applications that combine video, audio, text, graphics, etc. However, ATM networks lose some fraction of their input data cells because of overflowing network buffers, errors in cell headers, etc. The pattern of these losses is difficult to characterize since it varies greatly with the instantaneous bit-rates of the accepted traffic. However, losses tend to occur in bursts; if network buffers are full then some time must pass until they empty substantially.

Network designers have several tools that can be used to shape a channel's cell loss characteristics. Proper resource allocation and policing ensure that channels are not routinely swamped with more data than they can handle. Intelligent buffer management disciplines allow networks to protect important data from loss at the expense of more lost unprotected data. Forward error correction, data interleaving, and traffic shaping help reduce the severity of losses to the final receivers of transmitted data.

Because future broadband digital network models are still incompletely specified and because application designers have little experience with high-speed digital switched net-

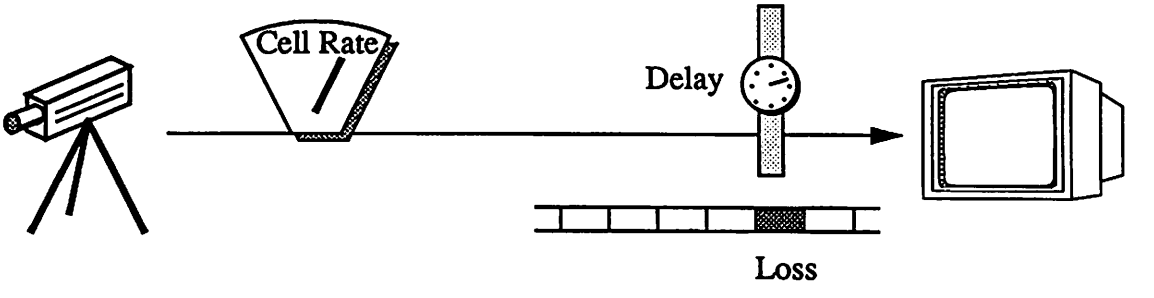
works, there remains a significant amount of work to be done integrating applications such as video communications into modern networks. One important problem addressed next is the specification of a more expressive signaling interface that allows networks and their client applications to communicate their capabilities and needs more effectively with each other than is possible with current models.

Chapter 3

MEDLEY INTERFACE PROPOSAL

The previous chapter concluded by presenting several new network interface models that support channel establishment, especially for multiway connections. This chapter presents a new signaling interface model, the Medley Interface, that extends previous work in several ways. The goal of this interface is to enable networks to implement channels with guaranteed quality of service (QOS) performance more efficiently than now possible. This is achieved partly through more detailed descriptions of applications' rate, delay, and loss needs than has been proposed. To facilitate this description, channels are decomposed into components called *substreams*, for which individualized descriptions can be specified. By describing substream characteristics rather than the characteristics of entire channels, applications avoid requesting channels with better QOS than they need.

A channel's *flowspec* defines its data rate, delay, and loss characteristics. The Medley



Channel Characteristics Described by a Flowspec (Fig. 7)

Interface defines a substream flowspec format that guarantees QOS performance while still allowing efficient network resource use. Many existing flowspec definitions are poorly defined, or they are so simple that they force networks to over-allocate resources to channels.

The Medley Interface describes a substream's cell loss characteristics with a new format that specifies the spacing and burstiness of cell losses in addition to the average cell loss rate. The description format must balance several trade-offs. In general as the format

becomes more powerful and complex, it becomes more likely to be able to support the transport needs of future broadband applications. The cost of this complexity is that call set-up negotiations, application design, and network design become more complex.

This chapter next presents a channel setup negotiation heuristic that allows applications to obtain minimal-cost channels that support a fixed level of application performance. Example negotiations are presented for two types of applications, and the implementation difficulties that arise are discussed.

The substream decomposition, new flowspec format, and channel negotiation technique cooperate to enable efficient network operation. The first two components enable accurate resource allocation and configuration of network components such as rate monitors and switches. Further, they enable applications and networks to gain advantage during channel setup negotiations, which reduce channel resource use directly. For negotiations to achieve significant resource savings, applications and networks must know enough about each other's sensitivities to different channel characteristics to trade among characteristics intelligently.

3.1 A Flexible Network Interface

In thinking about data communications systems, engineers often use the abstraction of the "ideal lossless channel" to try to decouple data transmission issues from the data encoding. In practice, this decoupling is often incomplete. Traditionally, since only one network has been available to application providers, the application designers have assumed one particular network behavior and then compensated for that fixed behavior within their applications. For example, the U.S. digital HDTV proposals all specify Reed-Solomon coding and ghost-cancellation methods to work well with through-the-air channels, even though HDTV probably will be broadcast over other media also.

Conversely, before integrated networks, most networks only supported a single type of application. As a result, the networks sometimes built in signal-processing functions to try to simplify their supported applications. For example, telephone networks provide echo

cancellers that are designed for the needs of voice conversations but are less appropriate for data modems or fax traffic. Also, cable television networks are well-suited for broadcast video distribution but do not allow for even the low-rate signaling required for applications such as pay-per-view video.

On the surface, it appears that the concept of the BISDN couples data sources and the network more tightly together. After all, applications that use the BISDN will have to negotiate QOS parameters and account for network delays and losses. However, network clients always have had to allow for network shortcomings; in the past, they could not choose the shortcomings. For example, the concept of the fixed-rate video codec is based not on any fixed entropy of the video source but on the need to transmit fixed-rate data over circuit-switched networks. What the Broadband ISDN does, rather than to force applications to make more allowances for non-ideal network characteristics, is to give them a much broader *choice* of network characteristics. Before every connection, applications can select channels with different delay bounds, different loss rates, different costs, etc.

One problem with proposed BISDN application-network interfaces is that applications are offered a very limited menu of transport services from the network. It currently is proposed that the BISDN define an application's needs fairly coarsely, in terms of only a few parameters such as average data rate, worst-case delay bound, and average loss rate. Also, the BISDN defines channels as if all of the data transmitted on each channel consists of one or at most two types—high loss priority and low loss priority. As a result, applications often are forced to transmit data over channels with inappropriate characteristics.

We propose that future broadband networks offer applications a much richer channel establishment interface, with more description of applications' transport needs than is available with a single priority bit. The interface should support flexible call setup negotiations so that applications and networks can establish channels with minimum-cost resource use. In this chapter we design and present a new interface model, called the Medley Interface, first to learn about the issues relevant in the design of such a model and also to

demonstrate the feasibility of the flexible interface concept.

The establishment of a communications channel through a network consists of several steps:

- the establishment of channel rate and QOS parameters
- the establishment of a route through the network over which the channel's data will pass
- the reservation of resources along the route to ensure that the channel's QOS specification will not be violated
- the proper configuration of network components in order to provide the channel most efficiently
- the proper configuration of application components to utilize the channel most efficiently

The Medley Interface proposal addresses the first of these steps. This chapter discusses how the Medley Interface model describes channel characteristics and how it implements call setup negotiations in order to arrive at a set of characteristics that is useful both for the communications application as well as the network it uses.

An important goal of the Medley Interface model is to manage complexity through the separation of the description of a transport service from the transport's implementation. Different pieces of network equipment could implement the same transmission channel in quite different ways. As faster switches, more elaborate buffer management disciplines, and more exact resource allocation formulae are developed, they can be incorporated into networks without affecting the network interfaces or existing applications.

The Medley Interface model also must be extensible. As newer versions of the interface are developed and integrated into existing systems, older networks and applications should continue to be able to negotiate and establish channels successfully. If a network interface supports options that an application does not understand or know about, the application simply can ignore those options. Newer networks should continue to be able to support older options by implementing the old options in terms of newer ones. For example, a new net-

work interface that prefers to use exponential-weighting rate monitors could continue to support leaky bucket rate monitors. The network simply would approximate the leaky bucket monitor with whatever process it uses to implement the exponential-weighting monitor.

This model proposes a change in the way that communications applications conceptualize networks—application designers are used to knowing a set of channel characteristics that they assume are unchangeable and that they try to design around. Now, applications designers must think about what QOS their applications actually need, how somewhat differently performing channels can be used to support the same application, and how an application's data traffic can be divided into subsets with different transmission needs. In this way, all of an application's traffic can be transported as efficiently as possible. Also, if an application requests a substream with a particular QOS that is temporarily not available, the application possibly will be able to alter its signal processing strategy to function with other available substreams.

The Medley Interface proposal actually decouples interactions between data sources and networks to some extent. For example, some applications currently perform error-correction coding on their data before it is sent to the network. By doing so, the applications essentially estimate the QOS of the network channel themselves and take their own, possibly quite inefficient, steps to modify the channel. A fairly modern error-correction code such as a Reed-Solomon code causes a marked increase in the amount of data sent over the channel. The application really just wants some level of network performance—if the network itself decides that error-correction coding or cell retransmission or other methods are appropriate to maintain that performance level, then the network can provide one of these functions.

Traffic shaping is another example of an application's attempt to modify the available transmission channel. The network itself can shape an application's traffic (subject to the application's delay requirements) more efficiently than the application itself. The network

memory can be shared among multiple applications, the network knows when traffic shaping is unnecessary or detrimental, etc. If possible, the network should handle these network functions, not the individual sources.

It is important that applications be accurate in requesting their desired delay and loss characteristics. Although zero-delay lossless channels would be nice for every application, such channels would and should be very expensive. Of course, if all applications were to use channels with higher qualities than they really needed, the network would not be able to support as many customers. If applications were forced to accept channels with too poor a QOS specification, then the applications themselves would try to implement network-recovery operations that would be more intelligently and efficiently done by the network. It will be a difficult and important problem in the future for network providers to set prices for their offered transport services so that client applications select channels that are neither too high-quality or inadequate for their needs.

If networks were capable of dynamically varying a channel's QOS specification, they could offer similar benefits to the proposed Medley Interface networks. However, dynamic QOS specification is difficult to implement in practice because it implies dynamic resource allocation. If resources were not available when a connection requires them, then the connection would cease to be useful. Dynamic resource allocation schemes thus must be very conservative, and thus inefficient. With multiple logical channels, the network can figure out the maximum resources needed by a given connection (i.e. set of substreams) during call setup. The network also can figure out the percentage of time that resources will be needed, given each substream's source description. Then, the network can allocate its resources efficiently.

Also, communications applications could obtain some of the benefits of the Medley Interface by purchasing multiple virtual circuits from a network. By using multiple virtual circuits, each with different qualities of service, an application could transmit different data types with different network qualities. In fact, the Medley Interface may implement differ-

ent substreams by establishing several virtual circuits. However, without the Medley Interface, communications applications themselves must be able to translate their transport needs into QOS parameters, and to map their data substreams onto various virtual circuits. Further, all of the benefits of the Medley Interface model are not available to applications that establish multiple virtual circuits. For example, it is not possible for an application to establish a virtual circuit with a quality of service dependent on the QOS of another virtual circuit.

The following parts of this chapter discuss the components of the Medley Interface in more detail, implementation of the model, and use of the Medley Interface model by applications and networks.

3.2 Transmission Channel Model: Substreams

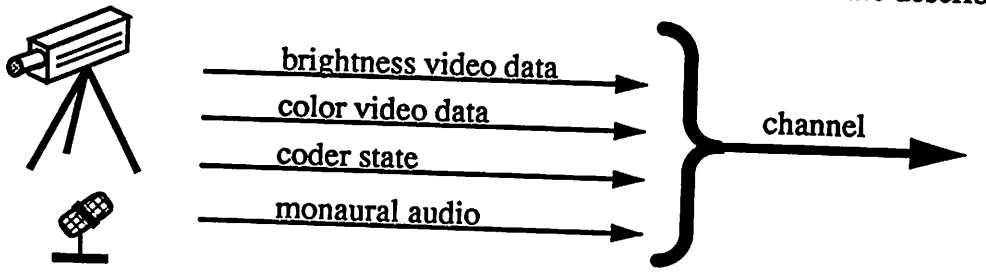
Communications applications often transmit a variety of different data types between network end-points. For example, a multimedia conferencing system might transmit still images, compressed motion video, compressed audio, text, graphics, and formatting information. Each of these data types has unique rate characteristics which depend upon the source material and the encoding technique. Further, delays and losses of each data type affect the perceived display quality differently. If all data types were transmitted over a single channel with homogeneous transmission parameters, then the channel would have to be configured for the most stringent flowspec requirements. Those requirements might vary considerably over all of the application's data types, so this approach could be quite inefficient. On the other hand, if separate channels were used to transmit each data type, then interdependencies between the QOS needs of the separate data types could not be expressed or utilized. Also, network management operations which only should be performed once per connection would need to be performed once for each data type's channel.

Traditionally, networks treat all data within a single channel homogeneously. The priority bit in the BISDN allows networks to identify two separate data streams within a single virtual circuit and to process those two streams differently. The Medley Interface proposes

that a layer of abstraction be placed between the virtual circuit and the application. This layer describes the transmission resources used by an application as consisting of a single channel that contains an arbitrary number of *substreams*. A BISDN network implements this channel with one or more virtual circuits. Each individual substream that belongs to a single channel can carry traffic with its own rate description and QOS specification. The application considers the types of data it needs to send, and the flowspec characteristics of those data, and decides itself how many substreams to request. While a compressed multimedia conferencing application may use numerous substreams, an uncompressed voice-only call may only use one.

Each substream in a channel can be described independently with its own bandwidth, delay, and loss requirements. Since the Medley Interface allows more detailed rate and QOS descriptions than do current BISDN proposals, it enables greater network efficiency than currently is possible. With the detailed substream description, the network can do a better job of routing, resource allocation, and buffer management than it could with only a general description of an application's rate and QOS needs. Importantly, the joint characteristics of several substreams also can be expressed within the Medley Interface model. For example, an application could specify different loss priority levels for each of its substreams.

Figure 8 shows how a combined audio-videoconferencing service could use multiple substreams for different data types. Each data type's rate characteristics are described sep-



Multiple Audio and Video Substreams in One Channel (Fig. 8)

arately. Audio data is sent over substreams with stringent loss bounds, while video data is

sent on substreams that tolerate more loss. Coder state information can tolerate no cell losses but can be delayed more than other data types. The substream decomposition allows all of these transport needs to be described separately, and allows a network to implement a channel that meets all of these needs exactly rather than meeting the most stringent requirements of all of the data types.

The substream model provides a convenient format in which applications can describe their rate characteristics and QOS needs, and it is independent of the network's implementation of the transport. When a network implements a Medley Interface channel, the network need not assign separate virtual circuits to each substream. The network could choose to aggregate multiple substreams on one virtual circuit, ensuring all substreams' QOS guarantees are met with proper multiplexing, FEC, priorities, and buffer management. Because of the separation of transport description and implementation, the network can use different implementations at different times to always use resources most efficiently.

The substream concept also increases network efficiency by allowing some network functions such as billing and security to be performed once per connection rather than once per substream or virtual circuit. These management operations are separated from data transport operations, and thus can be made less of a burden during call establishment and termination.

A goal of the Medley Interface is to support future as well as current applications and networks. Today's applications may only request one or two substreams per connection. Current networks may be constrained to implement all connections with just a single virtual circuit. However, future multimedia and multiway conferencing applications may need to transmit a wide variety of data types, and future networks likely will contain enough intelligence to route or buffer different substreams separately. Only a description format that supports a great deal more flexibility than is needed today can continue to be useful in the future. The description format for the multiple data types should not impede the development of new applications and networks.

The ability to describe transmission needs in detail should help change the way in which communications applications are designed. Without this ability an application designer unknowingly may construct communications algorithms that are tailored for the characteristics of a particular network. When the application is used on a different network or when the network characteristics change, the application would need retuning for optimum performance. With the Medley Interface, designers can concern themselves with the inherent communications needs of their applications. Further, when an application can trade between different QOS parameters, the Medley Interface frees applications designers to explore and select channels with a wide variety of characteristics.

3.3 Flowspec Definition Requirements

As discussed above, the decomposition of a channel into multiple substreams allows each substream's flowspec to be specified separately. Applications use their substream flowspecs to configure their data encoding methods to the channel characteristics and to guarantee their performance level to end-users; networks use flowspecs to set up channels with guaranteed QOS: a channel's route, resource reservation, switch configuration, and rate monitoring all depend on its flowspec parameters.

The definition of a flowspec format should allow applications to ensure that they provide adequate performance to their end-users and should allow networks to perform channel admission, rate monitoring, and resource allocation efficiently. Further, a good flowspec format should be detailed enough to enable channel setup negotiations to achieve significant network resource savings. Although many past works have presented flowspec formats, these works tend to be incomplete, inadequately specified, or too simple to enable efficient network operation. This section discusses *requirements* for a good flowspec format, and section 3.4 presents the Medley Interface flowspec format that meets these requirements.

3.3.1 Flowspec Must Be Well-Defined

In some past research works, QOS bounds are not useful because they are poorly or inadequately defined. For example several works describe channels' cell loss characteristics statistically, as a maximum probability of cell loss [48, 49, 52, 54, 56, 73, 76, 80]. Probabilistic bounds allow a network to claim that although the *probability* of cell loss for each of an application's 1000 offered cells is 0.01%, it happens that all of the application's cells were lost. This type of bound does not allow an application to rely on the quality of its channel since the meaning of the bound is not well-defined. Similarly, average bounds, such as on the average delay of a channel, often are not defined specifically enough. An average delay bound does not tell an application how long particular cells will be delayed or how many cells will exceed the given bound. For a bound to be useful it must specify hard numerical limits on the characteristics of the source or channel it describes, and it must specify the time interval over which the bound is valid. An example of a meaningful bound is, "On substream *A*, fewer than twenty cells will be lost in every time period of duration one second or shorter."

The ST-II protocol flowspec lets applications specify both their minimum and "desired" rates [80]. However, without more definition of how the desired rate parameter is treated, applications cannot rely on sending data faster than the minimum rate; the desired rate parameter does not guarantee channel QOS.

3.3.2 Flowspec Must Guarantee Channel Characteristics

For applications to guarantee their performance level, they should not have to operate with channels with poorer QOS than was established during channel setup. Thus, networks should allocate sufficient resources and processing to *guarantee* that negotiated QOS specifications actually will be provided. Without guaranteed QOS, application designers must build their applications based on *estimates* of the QOS that their applications will receive. The less accurate the estimates, the less efficient the applications and networks. If a network guarantees that established QOS specifications never will be violated, then applica-

tions designers can be confident of providing good performance to their end-users even with QOS specifications close to their minimum acceptable levels. Networks guarantee QOS specifications with appropriate channel admission and resource allocation policies; the design of efficient policies for different networks is a difficult open problem.

QOS guarantees also must be verifiable by a single application, so that applications can monitor a network's compliance with their guarantees. For example, guarantees that apply to a group of applications do not let individual applications rely on their negotiated bounds. Aggregate guarantees are not enforceable, and they again require applications to try to estimate the QOS they actually receive.

3.3.3 Flowspec Guarantees are Time-Invariant

In the absence of renegotiation of QOS parameters after channel establishment, QOS specifications should not vary with time. Non-time-invariant loss bounds or bounds that specify the treatment of individual cells constrain network implementations too severely; the network must monitor the treatment of individual cells rather than of aggregate traffic. Further, these bounds would be more difficult to describe than time-invariant bounds. Finally, time-varying bounds require that networks be able to vary their switch configurations and resource allocations dynamically. This creates the problem of how networks can support channels whose dynamic resource needs cannot be met after the channels already have been established.

3.3.4 Flowspec Enforces Time-Local Guarantees

Although QOS bounds should be time-invariant, they should enforce some degree of time-locality. We have studied the performance levels of several types of applications as they vary with their channel rate, delay, and loss characteristics. For example, video and audio applications can provide good perceived subjective performance with lossy channels as long as the number of cell losses that occur in given-length time intervals is controlled (section 4.5). Many video and audio applications can estimate lost data at their receivers in

order to display high-quality output in spite of cell losses. Common estimation techniques use signal values adjacent to lost data to estimate the lost data's original value. If cell losses occur in long bursts, then large contiguous data blocks are not available at the receiver; these blocks cannot be estimated. Even without lost data estimation, video sequences subjectively appear better if errors are small and widely spread than if they occur in large connected regions. A good flowspec format should be able to express this type of application requirement.

Other applications must resynchronize their transmitter and receiver after any cell loss, either a single loss or a large burst of losses. For example, if a packet-based file-transfer application source receives no packet acknowledgments within some time interval, the source resets and begins to retransmit previous packets (section 3.5.3). A low-rate voice communications system could reset by forcing both its transmitter and receiver to set their adaptive filters or voice-generator models to the same states. For these applications to function well, bursts of cell losses can be large but they must occur infrequently. A good flowspec format should be able to specify this behavior also.

Some proposed rate bounds specify the maximum number of cells that a source can send within a specified time interval [49]; loss bounds also should specify the number of cell losses that a network can allow in a described time interval. Time-local bounds are crucial to applications that benefit from control the spacing between groups of cell losses. Per-call bounds are not very useful; without knowing the duration of a call beforehand a network cannot efficiently implement a channel with call-level bounds.

3.3.4.1 Dynamical Systems for Monitoring Time-Local Behavior

The Medley Interface rate and loss description formats monitor channel behavior over specified time periods. A general way to do the monitoring is with a dynamical system, that is, a system with inputs, outputs, and a *state*. At every timestep, the current state value and input value are used to generate the next state value and an output value. Dynamical systems are not the most general types of input-output system (because the output at any time

depends only on the state value and the input) but they can implement a wide range of behaviors and their simplicity often is useful. Differential equations and difference equations are examples of dynamical systems. The continuous-time system $output(t) = input(\sqrt{t})$ is not a dynamical system; if we know the state value at a given time t_0 and the input after t_0, we cannot compute the entire output after t_0 (unless the state stores the entire past input function).

We next consider several dynamical systems that can monitor the data rate or loss characteristics of a substream. By guaranteeing that these monitors will not be violated, networks and applications can use them to *specify* QOS and rate characteristics as well as to *monitor* them. The output functions of the dynamical systems are designed to indicate whenever the systems' implied bounds are violated. The systems' *state transition functions*, the rules that the systems use to map the current state and input to the next state, are designed to provide meaningful monitoring of rate or loss characteristics.

Example:

> Suppose a dynamical system's state value at time n+1 equals the state value at time n plus the number of cell losses on substream X between times n and n+1, and the system's output equals its state value. This system measures the number of cell losses that occur on substream X but gives no information about the rate of losses.
>
> Suppose a second dynamical system has the same state transition function as the system above, and its output value is its state value divided by the time. This system gives a well-defined long-term average cell loss rate but does not give any information about cell loss characteristics over smaller time periods.

The space of possible dynamical systems is huge, but there is not much reason to consider any but the simplest. For example, non-time-invariant dynamical systems such as the second example above are not of much use in specifying behavior over limited time durations. Complicated systems such as

$$state(n+1) = \max(state(n)^3, 2 * state(n)^2 + 6)$$

also are of little use in measuring the rate of events. Some simple dynamical systems are

discussed below.

3.3.4.2 Leaky Buckets

The leaky bucket used for rate policing also can be used to monitor cell loss behavior. The leaky bucket is a dynamical system with the following state transition rule:

$$state(0) = 0$$

$$state(n+1) = \max(state(n) + events(n) - tokens(n), 0)$$

Events can be cell deliveries to the network or cell losses, depending on the type of flowspec bound being implemented. The output of the leaky bucket system is 1 if the state value exceeds L (the leaky bucket "size") and is 0 otherwise. (Some researchers specify that a leaky bucket's state value cannot rise above L [39].) *Tokens*, which decrement the leaky bucket state, are sent to the system either periodically or dependent upon a substream's traffic. If events enter the leaky bucket system faster than tokens, then the state value climbs towards L. Token arrivals drive the state value down towards 0 (the state value cannot fall below 0 however). Thus even if no events take place for a long time and then a burst of L or more events ever happens too quickly, the leaky bucket system will detect the violation.

The parameters of the leaky bucket monitor can be varied to implement bounds with different behavior. For example, a leaky bucket loss bound with $L = 0$ always forbids any cell loss. As the bucket size increases, the leaky bucket system allows longer and longer bursts of events to occur without signaling a violation. As the token arrival rate increases, the system allows events to occur more frequently.

One nice property of the leaky bucket system is that it is quite easy to implement. Applications could implement leaky bucket monitors to verify that the bounds that they were guaranteed by the network actually are being met. Networks could implement actual leaky buckets to monitor a source's rate or to verify their delivered QOS and to change their behavior when the QOS begins to deteriorate.

3.3.4.3 Time Windows

Flowspec bounds also could be described in terms of events per time-window. A dy-

namical system based on sliding time windows takes the following form, where

$$s(n) = \sum_{k=n-N}^{n} e(k)$$

is the system state, *e(n)* denotes the events being monitored, and *N* is the window size. The output of this system is 0 whenever the state is less than some limit *L* and 1 otherwise. An alternative window-based description system is based on non-overlapping windows.

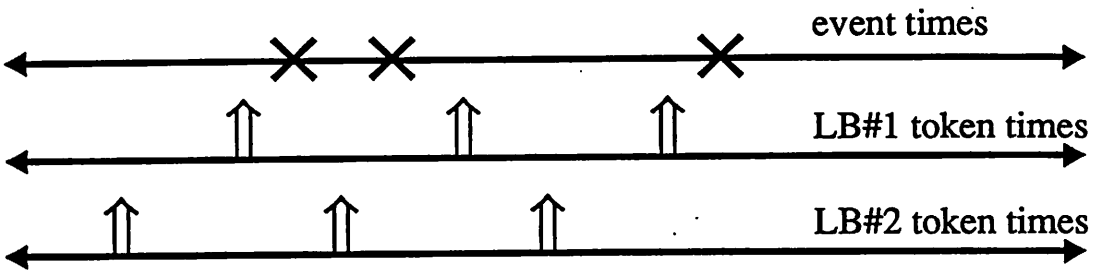
$$s(n) = \sum_{k=Kint(\frac{n}{K})}^{n} e(k)$$

The sliding-window based system's state at all times is the sum of the number of events during the preceding *N* seconds. The system based on non-overlapping windows defines windows that begin at times 0, *K*, 2*K*, etc. At all times, the state value is the number of events from the most recent window-start time until the current time. Of course, it also would be possible to implement time-window bounds that use durations defined in terms of substream traffic rather than time.

Time-window based rate and QOS monitors work similarly to leaky bucket monitors. A time-window's parameters, its size and its loss threshold, can be varied to implement different monitoring behavior. Large threshold sizes allow larger bursts of events to occur, and larger ratios of threshold to window size allow higher average event rates.

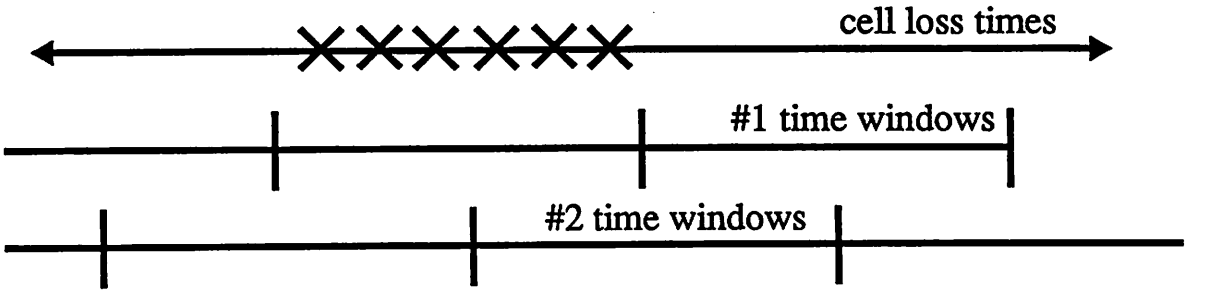
Any time-invariant dynamical system that measures the *rate* of events (including the leaky bucket and time-window systems) must decrement or reset its state value at some times. There can be ambiguities in the specification of the reset times with respect to the input times. For example, suppose a substream's loss characteristics is governed by a leaky bucket loss monitor that is fed a token every millisecond. Further suppose that both the network and the substream receiver both actually implement leaky bucket monitors to verify

the substream's QOS.



Different Leaky Bucket Monitors (Fig. 9)

Leaky bucket #1's state climbs to 2, but leaky bucket #2's state value only reaches 1. If the leaky bucket bound specified a maximum size of 1, then one bucket would detect a violation while the other one would not. The situation is worse with other monitors such as non-overlapping time-windows.



Different Time-Window Monitors (Fig. 10)

Here, time window monitor #1 detects six losses per window while time window #2 only detects three.

It is difficult to synchronize, or even to specify, exact start times for multiple monitors distributed throughout a network. Without synchronization, two monitors of the same process could detect different characteristics. For the leaky bucket system, we can prove that any two monitors that are identical except for the phases of their token times never differ in state value by more than 1. This fact argues that monitors based on leaky buckets may be more useful than those based on non-overlapping time windows for example.

Theorem: The state values of any two leaky bucket monitors that have the same token rate and that are fed the same event processes differ by at most 1. Their token arrivals may have different time phases.

Assume: Two leaky bucket monitors A and B both have periodic token arrivals at the same rate but different token time phases. A and B both monitor the same event process, and both start with state values equal to 0.

Proof: Consider a time $T1$ when both buckets have state values equal to 0. No losses can have occurred between the two most recent token arrival times at the two buckets, or else the bucket with the less recent token arrival would not have state value 0. After $T1$, losses occur that drive both state values above 0. At all times after $T1$ as long as both state values are greater than 0, the state value of each bucket equals the number of losses on the substream that occurred after $T1$ minus the number of tokens that have arrived after $T1$. At all such times, the numbers of tokens that have arrived at the two buckets differ by at most 1, since tokens arrive at the same rate. Define the time $T3$ to be the first time after $T1$ when a token arrives at a bucket (without loss of generality, bucket B) and the state value of the other bucket (bucket A) is 0. If no finite $T3$ exists, then the above argument explains why the state values of A and B never differ by more than 1.

Assuming a finite $T3$ exists, define $T2$ to be the oldest token arrival time at bucket A before $T3$. No losses can have occurred between $T2$ and $T3$ or else bucket A's state value would not be 0 at $T3$.

Right before $T2$ both state values must have been 1.

Lemma: A's state value must have been 1: If A's state value were greater than 1, then A's state value could not be 0 after $T2$. If A's state value were 0, then no losses could have occurred between the *previous* token arrival at bucket A and the arrival that defines $T2$. There must have been an arrival at bucket B between these two token arrivals at A, and *that* time would have met the conditions for the definition of $T3$. This contradicts the fact that our $T3$ is the earliest such time.

Lemma: B's state value must have been 1: if B's state value were 0, then the arrival at A that defines $T2$ would suit the conditions of the definition of $T3$ (with the roles of buckets A and B reversed). This contradicts the fact that $T3$ is the earliest such time. If B's state value were larger than 1, then at time $T2$, bucket A would have had two or more token arrivals since $T1$ than bucket B. This contradicts the fact that tokens arrive at the buckets at the same rate.

So, both buckets had state values equal to 1 right before $T2$, and no losses occurred between $T2$ and $T3$. Thus, both buckets have state value 0 at $T3$. Thus for all intervals between times in which both buckets' state values are 0, their state values differ by at most 1. Thus, at all times the two buckets' state values differ by at most 1.

The above proof argues that the leaky bucket system would be more useful for QOS specification and monitoring than a system based on nonoverlapping windows. Sliding window systems, since they monitor events over *all* time windows of a specified duration, do not suffer from synchronization ambiguities. However, sliding window systems are much more difficult to implement than leaky bucket systems since they must store the times of the most recent L events, where L is the allowed number of events per window. Leaky bucket systems can be implemented with a single counter.

3.3.4.4 Exponential Averages

Still other dynamical systems could be designed for the specification of time-limited channel characteristics. The exponential average system behaves as follows. The parameter

$$s(n+1) = \alpha s(n) + e(n)$$

α is a decay rate that determines how much of an effect past events have on the current system state, and $e(n)$ represents the events. As with the systems discussed above, an exponential average system outputs 0 if the system state is less than a given threshold L and outputs 1 if the system state is greater than L. Typically α is chosen in the range (0, 1). For α values close to 0 the system state decays to 0 quickly so past events do not affect the state very much. If α is closer to 1 then the state value decays more slowly and the dynamical system displays a longer memory.

The exponential system displays the same synchronization problem as the nonoverlapping time window system. If two otherwise-identical exponential systems start monitoring events on the same substream at different times, the state values will not always match. Depending on the value of α that the systems use the disparity may be large.

The exponential system is not much more difficult to implement than the leaky bucket system, although it does require high-speed multiplication with several bits of precision.

However, the exponential system does not map well to applications' needs. For example, with the exponential system it would be difficult to describe the application requirement that consecutive cells not be lost. This is more easily expressed with the leaky bucket or time-window systems.

3.3.5 Define Flowspec at an Appropriate Level of Detail

During call setup, an application can send a channel description to its network and ask the cost of the channel. If the cost were unacceptably high, the application could close the channel and request another with less demanding characteristics. This process could continue until the application guesses a channel description that provides an acceptable trade-off between the application's delivered performance and the channel's cost. Rather than relying on this hit-or-miss process for the negotiation of channel parameters, the Medley Interface proposes a formal negotiation procedure that tries to achieve the minimum cost channel possible for a fixed level of application performance.

The need to negotiate channel characteristics creates a trade-off in the design of a flowspec format. More powerful descriptions allow more exact rate and QOS specification. Additional information only can help networks more accurately allocate resources and configure themselves; networks that wish to ignore detailed QOS specifications such as multiple leaky bucket rate bounds can use only simpler ones. However, more complex flowspec formats make it more difficult to establish an intelligent negotiation procedure.

Flowspec descriptions could be arbitrarily complicated.

Example:

Identify cell loss times as t_1, t_2, ... Ensure that for all even i,

$$(t_{i-2}) < M * \exp(\alpha t_{i-1} * i) * \mathrm{sqrt}(k t_i)$$

In the specification of a flowspec format however, we must trade between the expressiveness of the specification (how general the specification is) and the simplicity of the specification (how easy it is to use for negotiations, resource allocation, application configuration, etc.). To describe the variety of rate, loss, and delay characteristics that it can

provide, a network could give applications:

- one choice
- a set of fixed choices
- parameterized choices
- a complex description language

The first two possibilities clearly do not offer enough flexibility to support a diversity of modern communication applications. Assume a network allowed applications to choose their channels' flowspec characteristics through the selection of one of a few high-level primitives such as "Video()", "Audio()", or "Text()". Each of these primitives would imply fixed rate, delay, and loss characteristics on a substream. Although this approach is simple to use and implement, it is too inflexible even for current applications. Multimedia presentation systems could include different numbers of video components, each with different resolutions and requirements. The spectrum of video telecommunications applications, from low-bitrate videophones to high-quality conferencing systems requires channels with widely varying characteristics.

Some past works propose that all of an application's flowspec parameters should be expressed in a single parameter—the application's "equivalent bandwidth" [44, 52, 54, 62], which incorporates information on its rate, rate burstiness, and loss needs. These works discuss narrowly defined source models such as on-off sources [52, 54] or interrupted Poisson processes [62]. They ignore possible variations in delay requirements between different applications. More importantly, this approach prevents channel setup negotiations from taking advantage of different trade-offs among QOS parameters in applications and their networks.

The opposite flowspec definition approach is to design a complex language. Such a language would be flexible but difficult to use for call setup negotiations. During a negotiation, applications and networks must translate flowspec parameters quickly into application performance and network cost measures. Further, negotiating entities must be able to perturb

a flowspec so as to achieve a better trade-off between cost and performance. With complex flowspec formats both of these actions become slower and more difficult. Further, syntactically incorrect flowspecs must be detected and handled automatically during call setup negotiations, which is more difficult with more complex description formats.

To balance the trade-offs inherent in flowspec description, the Medley Interface description model uses simple parameterized statements. The meanings of statements and combinations of statements are defined, and networks can detect syntax errors fairly easily. Further, through the addition of new primitives to the language, the Medley Interface can be extended to support future network capabilities without altering the language syntax and structure.

3.3.6 Flowspec Specifies Both Intra- and Inter-Substream Characteristics

There is a trade-off between how tightly an application can specify QOS guarantees on multiple substreams and how much the application constrains a network implementation of a channel with multiple streams. It is reasonable to expect that a network will transmit all data on a single substream over the same route and through the same buffers. This allows characteristics within a single substream to be controlled relatively tightly. However to require that networks route and process identically all substreams that make up a channel restricts networks' implementation flexibility much more. Unless different substreams do pass through the same network components though, it would be difficult for a network to offer much control of the joint cell discard or delay characteristics of multiple substreams. To avoid requiring different substreams be routed and buffered together, flowspec description formats should specify fewer multi-substream characteristics than single substream characteristics. For example, a format might allow an application to request that no cells be lost on substream A until all available data on B is discarded or to ask that the long-term loss rates on substreams C and D be equal but may not allow an application to request that cell losses be interleaved on substreams A and B.

3.4 Medley Interface Flowspec Format

Past flowspec definitions frequently do not meet all of the requirements presented in section 3.3. These definitions often rely on poorly defined probabilistic or average bounds, or they are so simple that they hamper efficient network resource use. Since no full-scale high-speed cell-relay network implementations yet exist, no flowspec definition has been shown to perform well in practice. Next we discuss how past works have defined channel flowspec parameters and define the Medley Interface flowspec format. To describe the characteristics of an entire channel, the Medley Interface first describes each substream's characteristics and then tells how the substreams' characteristics should be linked.

3.4.1 Rate Specification

Applications with compressed input signals or random sources transmit a time-varying number of cells per second. The average rate of time-varying sources is often easy to calculate, but this information alone does not capture enough of the characteristics of these sources for networks to implement their channels properly.

Example:

> A source that always sends one cell per second to a queue that holds five cells and is served every 0.9 seconds never overflows the queue.
>
> A source that outputs a Poisson process with average rate one cell per second that feeds the same queue overflows the queue with a probability of about 6%.
>
> A source that cutputs six cells per second with probability 1/6 and zero cells per second with probability 5/6 loses cells with a probability of around 3%.

Some recent works have studied network resource allocation given a particular rate description format [33, 44, 46, 54, 55, 76, 83]. Each of these studies tries one of two approaches, both of which derive channel bandwidth allocations for bursty sources based on their rate description:

- Force a statistically simple but possibly inaccurate model onto sources, and analytically derive the bandwidth and buffer space required to achieve given average cell

loss rates. Such models include Gaussian bit-rate histograms and two-state Markov-modulated Poisson processes.

- Use more general rate descriptions and perform simulations to estimate the bandwidth and buffer space necessary to achieve given long-term average cell-loss rates.

The simulation approach can support a wider variety of traffic types more accurately than the simple analytical approximations, but it does complicate the resource allocation problem somewhat. If a network uses the simulation approach to map source rate descriptions to resource requirements, then it must store tables of simulation results rather than a compact analytic formula. However, the simulation approach currently seems necessary for the estimation of bandwidth needs in networks that support heterogeneous traffic mixes. No analyses have yet calculated resource needs for different traffic types that share the same network.

Further, for a given rate description and buffer allocation, no analyses have been able to calculate any cell loss characteristics other than average cell loss rates. Since the range of resource allocation problems that are soluble analytically is so small, it seems reasonable to rely on simulations for the calculation of a channel's resource needs given a rate description and detailed cell loss description. Tables of simulation results could be organized into "sub-tables"—for example one for applications that specify only long-term average cell loss rates, another for those that specify consecutive-loss characteristics, another for specification of lossless intervals, etc. A prototype of a network with such tables is described in section 3.5.3.

All of the rate description mechanisms presented in the previous chapter attempt to capture the rate variability of a source in a simple way. The leaky bucket monitor (sections 2.3.1 and 3.3.4.2) is one such method. If a bursty source is guaranteed not to overflow a leaky bucket monitor of a given capacity and decrement rate, then the capacity and decrement rate contain information about both the source average rate and burst distribution. If a source is described as not violating several leaky bucket monitors, then more is known

about the source's rate characteristics. Small leaky buckets served at fast rates can bound the high-speed burst characteristics of a source, and larger, more slowly served leaky buckets can bound a source's average data rate.

Other researchers suggest using the average cell rate, peak cell rate, and maximum burst duration to describe bursty sources. The definitions of "average" and "peak" frequently are incomplete ([48] is an exception). Still other works propose two-state models in which each state corresponds to a different cell delivery rate. These types of models are well-suited to applications that themselves operate in either of two states, one in which data is sent at high speed and another in which no or little transport occurs. Such applications include file-transfer and speech compressed with silence detection and elimination. However, these models cannot capture the rate dynamics of traffic sources whose output rate varies across a continuum, such as compressed video sources.

More exotic rate description formats exist as well. In [54], Guerin et. al. propose that a source's rate histogram be approximated by a Gaussian distribution (with moment matching). Using knowledge of the properties of the Gaussian, Guerin derives needed queue sizes for various probabilities of queue overflow. The Gaussian approximation is most valid for sums of large numbers of identically distributed sources (because of the central limit theorem) but is fairly inaccurate for combinations of small numbers of sources or for sources with widely varying characteristics.

The Medley Interface uses leaky bucket bounds to describe a substream's rate characteristics. A leaky bucket rate monitor is denoted RLB(A, M, N), where A identifies the substream being described, M is the maximum allowed leaky bucket state value, and N denotes the number of leaky bucket tokens generated per second. The behavior of leaky bucket rate monitors is well-defined, leaky bucket monitors with different token phases behave nearly identically, and leaky bucket monitors can specify a substream's burstiness at various rates through proper choice of the parameters M and N.

For simplicity, the Medley Interface allows only one or two monitors to describe each

substream's traffic. Two rate monitors both can be imposed simultaneously on a substream with the RUNION(x, y) primitive; both x and y are RLB() bounds.

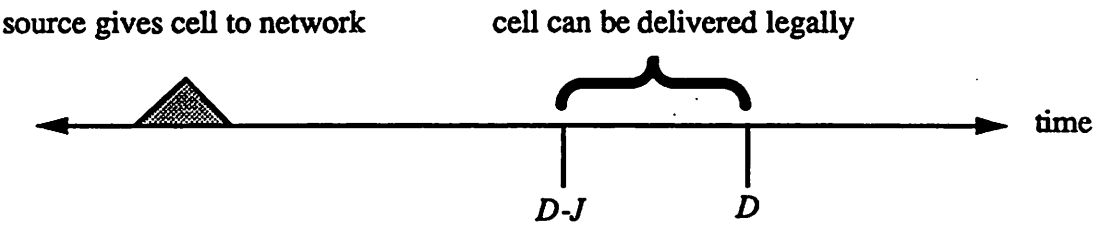
3.4.2 Delay Specification

The treatment of delay bounds throughout the literature has been nearly uniform [21, 49, 53, 60, 75, 80, 84]. Nearly all works that mention channel delay requirements at all specify a maximum allowed channel delay and optionally a *delay jitter*, or maximum allowed delay variation. The specification of channel jitter is equivalent to the specification of a minimum allowed channel delay. One work has presented a queueing discipline that attempts to minimize the expected value of a channel's delay [35].

We know of no efforts that have tried to capture any more complicated delay statistics within channels. This absence is not so crucial, however. Communications applications use delay bounds to guarantee their interactive response time. Perhaps more importantly, many applications use maximum delay and maximum delay jitter bounds to configure storage buffers at the application receiver. An application that continuously displays a signal (e.g. video, audio, graphics) at its receiver must be fed a steady stream of data. A buffer in the receiver smooths out delay variations in the application's channel and provides a jitter-free stream of data to be decoded and displayed. The receiver must not begin processing data from this buffer until a time at least equal to the channel delay has passed since the data was originally transmitted, or the receiver buffer could underflow. To prevent buffer overflow, the receiver buffer must have a capacity at least equal to the channel's peak rate multiplied by the channel's maximum jitter. Any larger buffer size never would be used, and any smaller size would overflow. Thus, a channel's maximum delay and maximum delay jitter are sufficient information for the application that uses it to provide interactive delay bounds and to dimension receive buffers. Additional delay information serves little purpose, and other delay measures are not as useful.

The Medley Interface specifies a substream's maximum allowed delay with the DELAY(A, D) primitive; A identifies the substream being monitored and D denotes the al-

lowed substream delay in seconds. A substream's maximum allowed delay jitter is specified with a JITTER(*A*, *J*) bound; *J* specifies the maximum jitter on substream *A*. Any cells on substream *A* delivered before *D*-*J* or after *D* seconds are considered lost.



Substream Jitter and Delay (Fig. 11)

Relative differences in delay on two or more substreams are constrained somewhat by the substreams' delay and jitter bounds. However, applications may desire tighter control—that cells on two different substreams leave the network in the same order as which they arrived. In-order delivery within a substream is guaranteed, but is not between separate substreams unless specified. The Medley Interface specifies that cells on two different substreams *A* and *B* should be delivered in-order with the SEQUENCE(*A*, *B*) primitive. Sequencing can be guaranteed on more than two substreams by combining SEQUENCE() bounds.

Example:

> SEQUENCE(*A*, *B*) + SEQUENCE(*A*, *C*) + SEQUENCE(*B*, *C*) guarantees that cells on substreams *A*, *B*, and *C* all are delivered in the same order as they were given to the network.

3.4.3 Loss Specification

The treatment of channel loss characteristics also has been fairly uniform in the recent literature [21, 35, 49, 54, 64, 73, 75]. All works describe a channel's loss characteristics through its average cell loss probability. None define "average" adequately however, by specifying a time-interval over which the average is computed, by defining the average to be computed at the time of channel tear-down, or by requiring that at every instant during the channel lifetime that the number of cell losses divided by the lifetime be less than the specified bound. The cited works also give no attention to the specification of cell loss char-

acteristics at short timescales.

The Medley Interface uses leaky buckets to monitor channels' time-local cell loss characteristics. If any one type of dynamical system exhibited analytic or practical superiority over others then there would be a clear choice of a best cell loss monitoring technique, but currently no overwhelming advantage exists. However, the leaky bucket monitor is simple to use and to implement, and it does not suffer much from time-synchronization problems between different versions of the same monitor.

The Medley Interface flowspec format denotes a leaky bucket loss monitor as LLB(A, L, N), where A indicates the substream being monitored, L is the maximum state value allowed by the leaky bucket without indicating an error, and N is the number of cells that must be delivered successfully on the substream to cause a decrement token to be sent to the leaky bucket. A single leaky bucket with a large L value can be used to monitor the long-term average cell loss rate of a substream. If the long-term average cell loss rate on the substream is greater than one loss per N delivered cells, then eventually the leaky bucket will detect an error. However, the fact that the bucket does not flag an error until the bucket state value reaches L allows bursts of losses to occur without violating the monitor. The larger is L the more tolerant of error bursts is the leaky bucket monitor.

Example:

> A substream B whose loss characteristics are bounded by an LLB(B, 20, 10^4) monitor cannot have a long-term average cell loss probability higher than 10^{-4} and cannot tolerate more than 20 cell losses within any time period in which fewer than 10^4 cells are delivered successfully.

We choose to send tokens to the leaky bucket based on cell deliveries rather than time intervals. This allows the same leaky bucket bound to be used to specify the spacing between consecutive losses by using leaky buckets that have small values for both L and N.

Example:

> A substream C with an LLB(C, 1, 1) bound placed upon it cannot allow two cells in a row to be lost. If the bound were LLB(C, 1, 2) then at most one cell could be lost from every set of three consecutive cells sent on

the substream. If the bound were LLB(C, 2, 2) then at most two cells could be lost from every set of four consecutive cells.

In general, an LLB(A, x, y) constraint requires that at most x cells be lost from any string of $x + y$ consecutive cells delivered to substream A.

An application might want to impose simultaneously more than one loss bound on a single substream. The Medley Interface expresses the combination of two bounds with the LUNION() keyword.

Example:

A substream with the bound LUNION(LLB(E, 20, 10^5), LLB(E, 1, 2)) imposed upon it has its cell loss burstiness rate specified at two different timescales.

To specify the occurrence of loss-free periods, as would be done by a file-transfer application, the Medley Interface uses a dynamical system that is a simplification of the leaky bucket called the *toggle system*. The toggle system state transition function behaves as follows

$$s(n+1) = \mathbf{1}(e(n))$$

The function $\mathbf{1}(e(n))$ is 1 if one or more cell losses occur during interval n and is 0 otherwise; the system is called a toggle system because its state value is either 0 or 1. A token, which is generated whenever a given number of cells are delivered on a substream, moves the system from time n to time n+1. This system is identical to a leaky bucket system in which the state value cannot increase above 1.

The toggle monitor is useful because whenever it detects a loss it remembers the loss for a given time period—the amount of time required for a token arrival. One or more losses affect the monitor state identically. This is the same type of behavior exhibited by communications applications that cannot recover from a cell loss until they reset!

The Medley Interface denotes a toggle monitor as TB(A, N); N is the number of cells that must be delivered on substream A to send a token to this monitor. Any given application can choose N to match its own reset behavior. The toggle monitor alone does not im-

plement a complete cell loss burstiness bound, however. Any cell loss would cause the monitor to signal a violation. What we would like is to specify the rate at which violations of the toggle monitor occur. This can be done with a leaky bucket that monitors violations of the toggle monitor! The Medley Interface denotes the combination of these monitors as LLB(TB(*A, N1*), *L, N2*).

Example:

> Suppose losses on substream *D* are bounded by LLB(TB(*D*, 30), 5, 1000). The substream can allow bursts of 1 to 30 cell losses to occur at an average rate of at most once per 1000 successfully delivered cells. Up to 5 such bursts can occur sporadically per 1000 successfully delivered cells without violating the leaky bucket monitor.

With just the constructs described above, the Medley Interface can describe cell loss characteristics within a substream much more precisely than would be possible if it only specified the substream's average cell loss rate. The most common mechanism for the specification of related loss characteristics on multiple data streams is "loss priority", a somewhat vaguely defined concept that defines when cells on one stream should be lost in terms of losses on another stream. We define a substream *A* to have an *absolute loss priority* higher than substream *B* at a *particular buffer* if no cells on substream *A* are lost until all cells on substream *B* have been discarded from the buffer. A substream *C* is defined to have a higher *relative loss priority* than substream *D* if the long-term average cell loss rate of substream *C* is less than that of substream *D*.

Broadband ISDN channel specification proposals support both absolute and relative loss priorities, and any new cell loss description formats should do so also. BISDN proposals allow only two absolute priority levels per virtual circuit, however [104]. Two levels alone probably are inadequate for applications such as multimedia that use many different data types. A multimedia editing application could transmit more than a dozen different data types, each with a unique effect on the application's perceived performance.

The Medley Interface specifies absolute priority ordering among the several substreams

that make up a single channel with the ABSPRI(*substream, level*) primitive. Substreams have absolute priority level 0 by default, and the ABSPRI() directive lets any substream's priority level be set to any integral value above 0. If several substreams are given the same absolute priority level then no priority-based buffer discard relation exists among those substreams. If one substream has a higher priority level than another, then data on the higher-priority substream should not be lost whenever data on the lower priority substream can be discarded instead. The ABSPRI() directive is more of a hint to networks rather than a requirement. If a network routes two substreams with different priority levels through entirely different buffers and links, then the network probably cannot force losses on the high-priority substream to occur only after all data on the low-priority substream is lost. Within the Medley Interface, we choose not to allow the specification of absolute priority levels to constrain networks' channel implementations.

3.4.4 Summary

Below is the complete grammar for the Medley Interface flowspec format.

top-level	rate + delay + loss SEQUENCE(), ABSPRI() optional
rate	RUNION() or RLB()
RUNION(x, y)	$x, y \in$ RLB()
RLB(x, y, z)	$x \in$ substream ID $y, z \in Z^+$
delay	DELAY() or DELAY() + JITTER()
DELAY(x, y)	$x \in$ substream ID $y \in R^+$
JITTER(x, y)	$x \in$ substream ID $y \in R^+$
SEQUENCE(x, y)	$x, y \in$ substream ID
loss	LUNION() or LLB()

LUNION(x, y)	$x, y \in$ LLB()
LLB(x, y, z)	$x \in$ type $y, z \in Z^+$
type	substream ID or TB()
TB(x, y)	$x \in$ substream ID $y \in Z^+$
ABSPRI(x, y)	$x \in$ substream ID $y \in Z^+$

The syntax of this language could be extended to allow more complex bounds.

Example:

LUNION(LUNION(LLB(LLB(F, 5, 100), 10, 1000), LLB(F, 1, 3)),
TB(LLB(F, 100, 500), 10^4))

We choose not to allow any such extensions currently. For example, while many researchers have found it beneficial to control long-term average cell loss rate, and while we have found applications that operate more efficiently with control of simple loss burstiness characteristics, no one has shown the utility of controlling cell loss characteristics in more detail than specified with the current Medley Interface flowspec format.

It would be difficult for applications to decide when such complex bounds would be appropriate. It would be difficult for networks to decide what type of control to use to actually implement such a channel. Also, to exchange call setup information and negotiate flowspec parameters mutually agreeable to a network and an application would be more difficult with a more complex flowspec description format. However, more exact description formats may be beneficial in the future when new applications need exotic control of cell loss characteristics and when powerful network management methods can provide such control.

The simplicity of this language makes it fairly easy to interpret. No self-contradictory statements are possible in the language; however redundant statements are possible. Redundancy occurs when one component of a UNION() bound implies the other.

Example:

> In the bound RUNION(RLB(G, 5, 10^4), RLB(G, 10, 10^4)), the second component bound cannot be violated unless the first is also.
>
> The bound LUNION(LLB(H, 5, 500), LLB(H, 5, 1000)) behaves similarly.

The existence of redundant bounds should not pose any difficulty for networks—a network simply can provide a channel that meets the more stringent of the bounds. If bounds could be self-contradictory then networks would need to be able to identify the contradiction and inform the requesting application.

What remains to be seen is if this formulation for the expression of rate and QOS bounds is useful. The language must be powerful enough to express any practical channel characteristics that applications might need, and it must be easy enough to use that applications designers can take advantage of its capabilities and networks can implement channels that obey its bounds. Section 3.4.5 shows examples of how this language can be used by realistic applications.

3.4.5 Examples of Guarantees for Different Applications

This section shows Medley Interface flowspec bounds that would be used by a range of applications. Hopefully this helps convince readers that the format is flexible enough to support future application needs also. More detailed application simulations that use the Medley Interface flowspec format are described in sections 3.5.3, 3.5.4, and 4.5. The first examples here illustrate simple loss and loss burstiness bounds.

Examples:

> LLB(J, 0, 1) implies that substream J must be lossless.
>
> $\varnothing$ symbolizes no guarantees for a connection. A network should allocate no resources for this connection, thus providing traditional "best-effort" transport service.
>
> LLB(K, 10, 10^5) imposes a maximum loss rate within a time interval.
>
> LLB(L, 100, 10^5) implies the same average cell loss rate as above, but with a looser bound on loss burstiness.

Next, we show how a file-transfer application could describe its flowspec. All data within files to be transferred can be treated identically, so the application needs only one substream, called substream *A*. The application is not extremely delay-limited, so it can retransmit lost data. However, if data losses become too frequent then the retransmission delays become intolerable. Further, the application fragments large data packets into much smaller ATM cells before retransmission. If any cell in a packet is lost, the entire packet must be retransmitted. Thus, given a fixed cell loss rate, the application operates more efficiently if losses are tightly grouped together, because then fewer packets require retransmission.

The application transmits at a constant rate, say 10^5 cells per second, which it can describe with the statement RLB(*A*, 1, 10^5). The application's loose delay needs are expressed with the DELAY(*A*, 3) primitive.

The application can specify a worst-case long-term average cell loss rate with the statement

LLB(*A*, 100, 10^4)

The application does not really prefer correlated losses; it works best with long lossless periods, between which the channel can lose many cells. The application designer decides what channel characteristics are minimally acceptable and specifies them with a statement such as

LLB(TB(*A*, 20), 5, 1000)

Fewer than five times per thousand cell deliveries can the network allow twenty-cell-long bursts that contain losses. However, during those lossy bursts, the network can lose all of the data that the file transfer application delivers.

These two bounds can be combined with a LUNION bound.

LUNION(LLB(*A*, 100, 10^4), LLB(TB(*A*, 20), 5, 1000))

A video application's flowspec may be more complex. Suppose a video telephone uses intraframe coding, in which sequential frames are compressed and transmitted indepen-

dently. The videophone could use standard JPEG encoding [103], which uses the discrete cosine transform (DCT) to achieve most of its compression. The output coefficients from the DCT can be grouped into two sets—high-priority coefficients sent on substream *A* and low-priority coefficients sent on substream *B*.

Substreams *A* and *B* have known time-varying rate characteristics.

RUNION(RLB(*A*, 2, $2x10^4$), RLB(*A*, 25, 10^4))
RUNION(RLB(*B*, 4, $8x10^4$), RLB(*B*, 20, $3x10^4$))

Both have the same delay and jitter bounds.

DELAY(*A*, 0.2), JITTER(*A*, 0.15), DELAY(*B*, 0.2), JITTER(*B*, 0.15)

Substream *A* should have fewer losses than substream *B*.

ABSPRI(*A*, 1)

Both substreams have certain worst-case long-term average loss bounds.

LLB(*A*, 100, 10^7)
LLB(*B*, 100, 10^4)

Losses on substream *B* can be recovered through estimation if the loss bursts are not too large.

LLB(*B*, 2, 5)

Lost data on substream *A* cannot be recovered very effectively. However, if losses of substream *A*'s data are spread out rather than bunched then the decoded video subjectively looks better. There is a benefit in the limitation of loss burstiness on substream *A* also.

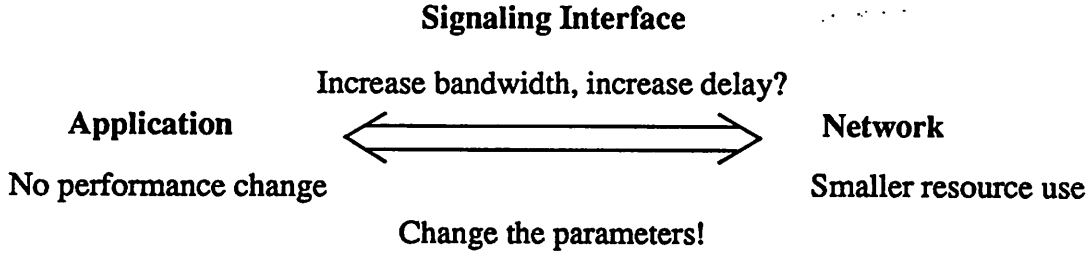
LLB(*A*, 1, 3)

Finally, each substream's two leaky bucket bounds can be combined within LUNION() bounds.

3.5 Call Setup Negotiation Protocol

The flowspec format described above can be used by communications applications to express their transport requirements. Most simply, a network would provide a channel that meets whatever loss, delay, and rate requirements that an application specifies. However, since an application could utilize a variety of different networks to provide data transport,

different channel flowspecs might provide the best combination of adequate application performance and low cost with each network. If call setup negotiations were allowed to be more complicated than the traditional "request-acknowledge-accept" cycle, then applications could obtain efficient and useful channels regardless of their underlying network (figure 12). Negotiations allow applications to compare the feasibility and cost of multiple flowspecs. A good negotiation strategy conducts this search quickly and reaches a flowspec that is optimal in some sense. For negotiations to be simple and rapid, the flowspec descrip-



Negotiations Save Network Resources (Fig. 12)

tion used during negotiation must be simple and easy to process; this need influenced some of the trade-offs made in the design of the Medley Interface flowspec format.

Some networks may not support all flowspec bounds allowed by the Medley Interface or by any other description format. For example, networks built to conform directly to the BISDN interface may not support the control of loss-free time intervals; networks should tell applications what flowspec parameters can be used. However, by separating the transport specification from its implementation, applications can take advantage of networks that do choose to support advanced capabilities. (A buffer control discipline that helps channels to guarantee the duration of loss-free intervals is presented in section 5.4.)

Similarly, some applications may not wish to negotiate all of the parameters available in the Medley Interface flowspec format. Many applications may not care about their channel jitter characteristics or may only use one leaky bucket rate monitor and one loss monitor. Applications must specify at least one rate bound, one loss bound, and one delay bound on each substream to allow networks to implement their channels, but applications need not

understand every part of the Medley Interface flowspec format in order to use it.

The Medley Interface flowspec *structure* allows new capabilities to be added to the format while it retains the same grammar, thus extending the format's power without requiring significant modifications to networks or applications. For example UNION() bounds could be allowed to contain other UNION() bounds. Parsers could be designed either to incorporate these new features easily or to detect and reject them. Again, only some networks need support a particular language extension. Network providers could compete with each other to support more extensions and thus a greater variety of transport services.

3.5.1 Optimal Negotiations

When communications application designers assumed only a single transport network, they could use their knowledge of the network to tune their applications for maximum efficiency. For example, the designers of the U. S. digital high-definition television proposals used their knowledge of through-the-air channel bandwidth and noise limitations when choosing their proposals' resolutions and error-correction methods [92, 94, 96, 98]. When applications utilize different transport networks at different times, a single set of flowspec parameters may be impossible or inefficient. To operate efficiently, applications and networks must exchange information about each other's needs and capabilities as part of a method that finds flowspec parameters that balance the application's QOS requirements with the need for network efficiency.

Network resource and management requirements for a channel with a given description are summed up conveniently by a *cost function c(x).* A channel's cost *c(x)* depends upon the characteristics of each substream that makes up the channel as well as on interactions between different substreams' specifications. These characteristics are embodied in a vector x of flowspec parameters; x also can be thought of as being a "point" in the space of flowspec parameters.

It would be nice if the costs of multiple substreams were additive. However, it is reasonable to expect that resource sharing between substreams will decrease a channel's cost,

and costs may be incurred for the establishment of special control of multiple substreams such as loss priority handling. These control costs are not local to a single substream and they are not proportional to the number of substreams that employ the control. To design a network's cost function $c(x)$, even for a fixed route, may be quite complicated.

In public networks such as the telephone system, real cost functions exist. They express, usually in dollars per minute of channel usage, the network effort required to provide a channel. A channel's cost is somewhat proportional to its resource requirements; the more link bandwidth and storage space required to implement the channel, the higher its cost. A channel's cost also may reflect processing resources devoted to the channel by the network. For example, channels that require complicated buffer management policies may cost more than channels that do not.

The reduction of all of a channel's resource and processing needs to a single number is a simplification, but it is a simplification that helps formalize what good call setup negotiations should try to accomplish. Even networks that do not bill their clients for their services can develop the concept of channel cost to reflect the channel's resource and processing needs. A cost function codifies the trade-offs among different resources within a network. For example, a network that uses undersea cables or scarce satellite links probably charges more for bandwidth as compared to buffer space than a smaller-area network that uses cheap transmission links. Of course, the cost that a network charges for a channel can depend on the channel's route. Channels that require many transmission links and that traverse many network switches consume more network resources and thus should cost more than shorter-distance channels.

Hopefully a channel's cost function does not vary over the duration of a call, or at least during call setup. Then, a network could maintain a description of its cost function at all entities that accept call setup requests. This would allow call setup negotiations to involve local negotiations only, between an application and the network interface to which it connects directly. If call setup negotiations were to require communications among a number

of widely separated network components, then the negotiation process itself would be slow and expensive. Once negotiations establish channel parameters and control, the network can reserve resources and set up control throughout the channel's route.

Every application must understand how the characteristics of its underlying channel affects the performance it delivers to its users. Just as networks combine several different resource and processing requirements for a single channel into a cost, it would be useful during negotiations if applications could map flowspec parameters into a single *performance function $p(x)$* that measures application performance as a function of its channel flowspec parameters. The design of application performance functions is more difficult than the design of cost functions, however. During the design of a cost function, a network designer knows what resources and processing are required for any given channel. The difficulty is to design a cost function that trades between channel needs intelligently. In the design of a performance function, the quantity being measured may be difficult to quantify. For example, video and audio systems often are compared subjectively, since no objective, calculable performance function has been found that adequately correlates with people's preferences. In subjective tests, human test subjects either compare the outputs of two systems and identify a preferred one or they give a subjective rating such as "good," "fair," or "poor" to different systems. With only this type of subjective test to rely on, to design a performance function that describes realistically performance differences that would result from different transport characteristics would be quite difficult.

The negotiation procedure described below uses the performance function $p(x)$ to calculate a *constant-performance plane* at each x. This plane is the linear surface about x in flowspec parameter space in which the application performance is constant; this is the plane perpendicular to the gradient of $p(x)$. Applications that cannot assign a numerical performance values to flowspec parameter points can use a simplified description of their constant-performance planes. Rather than assigning a performance value to every possible point x in flowspec parameter space, an application can define several *performance levels*

at which it operates. For each performance level K the application specifies a *performance level-set* P_K that contains all flowspec parameter points that support that performance level, i.e. $P_K = \{x: p(x) = K\}$. Local linear approximations of the performance level-set serve as constant-performance planes; an approximation method is discussed in section 3.5.4.

For example, a video-on-demand application could specify three performance levels: low, medium, and high. For each performance level, the application designer specifies the performance level-set P_K. The design of performance level-sets can be done with subjective testing fairly easily. A designer simulates the application with different channel parameter points and then has the subjective testers rate pairs of results as either equivalent in quality or not, and a set of channel parameter points that is judged to produce video of equivalent quality forms a performance level-set. We have found such tests to be easier to administer and to give more repeatable results than tests in which people are asked to assign numerical scores to the quality of different video sequences. An example of this testing procedure is discussed for a video application in section 3.5.4.

3.5.2 Negotiation Algorithm

To set up a channel, negotiations between an application and network must establish characteristics for each substream within the channel. During negotiations, the characteristics of the several substreams can be traded off against each other in order to achieve an optimal balance between application performance and channel cost. With the restriction that applications may specify performance level-sets P_K rather than performance functions $p(x)$, it is simpler to conduct negotiations that minimize cost at a fixed performance level than it would be to maximize performance at a fixed cost. That is, negotiations find the x that minimizes $c(x)$ subject to $p(x) = K$. Of course it would be possible for both channel cost and application performance to vary during negotiations. However, the trade-off between performance and cost depends upon the desires of the human users of a communications application; this is more of an economic or psychological issue than an engineering one.

Substream flowspec parameters such as maximum delay are easily represented numer-

ically. Rate and loss bounds both are represented with one or two leaky bucket monitors, each of which has two parameters that could be varied during negotiations. These characteristics are easy to vary gradually during a negotiation in order to explore different channel configurations.

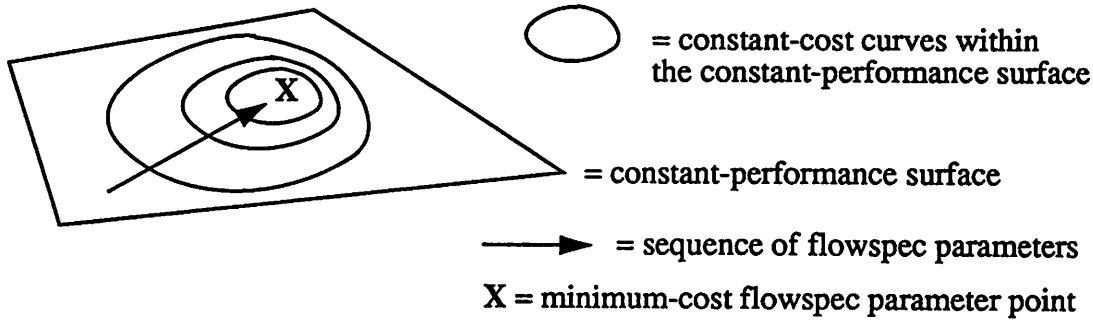
Other channel characteristics, such as the absence or presence of multiple levels of absolute loss priority, cannot be varied gradually during a negotiation. Generally, the channel characteristics that are not easily parameterized are the absence or presence of control capabilities. Rate, delay, and loss characteristics vary smoothly with the amount of resources allocated to a channel, and resources can be allocated with fine granularity. (For example, section 5.2.2 discusses how the partial buffer sharing discipline can adjust lost rates for multiple priority classes with fine granularity.) The provision of loss priorities among the substreams that make up a channel, of delay priorities, or of the ability to control cell loss spacing or burstiness all depend upon a network's processing capabilities. Negotiations must be able to identify these unsupported capabilities to applications; flexible applications should be able to operate without them. If these capabilities can be provided to a channel for free, then applications simply can request the capabilities that they find useful and negotiations can focus only on the aspects of channel description that affect resource allocation. However, if these capabilities do increase the channel cost, then negotiations must establish whether the controls are worth their cost. These negotiation decisions are of a different character than the decisions about how much delay or cell loss a substream will have, and this difficult problem is not studied in detail here.

Even given a complete specification of an application's performance function and a network's cost function, to find minimum-cost flowspec parameters for a fixed performance level is not straightforward. If both functions were strictly linear (or if one were linear and the other quadratic), then linear programming methods could be applied. However, the functions can be quite nonlinear. Even simpler mathematical problems such as the minimization of a non-quadratic function often must be solved using numerical iterative meth-

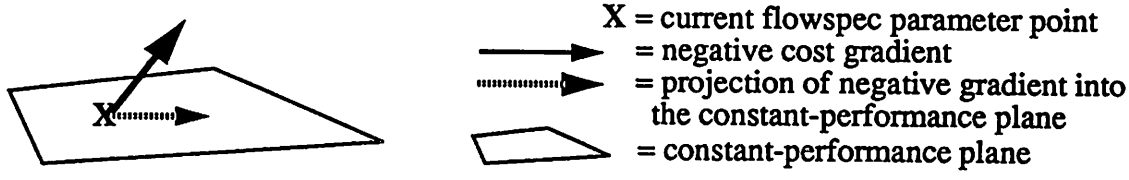
ods. Next we present an iterative method that tries to find the minimum-cost flowspec parameters that allow an application to maintain a specified performance level.

Iterative numerical techniques generally work by refining a candidate solution to a given problem repeatedly until it is as accurate as necessary. The cost-minimization technique discussed next uses this strategy. First, the application specifies a starting flowspec parameter point x_0 that yields the desired performance level. Repeatedly a point x_i is produced with a small update to x_{i-1} such that the performance of the application with the altered flowspec remains the same but the cost of the channel decreases. Eventually, this technique should reach a point for which any change results in a higher channel cost.

Each updated flowspec parameter point x_i must maintain a fixed application performance but must reduce channel cost. For each update to maintain a fixed performance, the updated points must remain within the previous point's constant-performance plane. For the updates to reduce cost most rapidly, they should be in the direction of the negative gradient of the cost function.



To achieve both of these goals simultaneously, we make iteration updates in the direction of the *projection* of the negative cost gradient onto the constant-performance plane.



The expression below calculates the projection of the negative cost gradient into the con-

stant-performance plane.

$$update_i = \rho\left(-\nabla c(x_i) - \frac{-\nabla c(x_i)\cdot\nabla p(x_i)}{\|\nabla p(x_i)\|_2^2}\nabla p(x_i)\right)$$

This negotiation technique is a modification of existing gradient descent techniques.

The projection rule determines the *direction* of the flowspec parameter point updates during iteration. The choice of the *size* of the updates is a trade-off. Larger steps allow fewer iterations but are less accurate. The projection operation assumes that the constant-performance plane is a perfect approximation to the constant-performance surface in flowspec parameter space. The larger the projection vector, the less accurate this assumption and the more the performance associated with the updated flowspec parameter point may vary. We choose step-sizes ρ_i dynamically to balance these trade-offs. At each iteration step, $x_{i+1} = x_i + \rho_i \cdot update_i$. After a step, we see how different are $p(x_{i+1})$ and $p(x_i)$. If this difference is smaller than some threshold then we increase the stepsize ρ slightly. If it is larger, we undo this most recent step and try smaller stepsizes until one is found that does not exceed our performance change threshold. Thus the stepsize gradually increases in regions of the channel parameter space in which the constant-performance surface is relatively linear with respect to the stepsize, but when a nonlinear region is entered, then the stepsize is reduced until the constant-performance surface again appears to be linear.

We terminate the iteration when the projections become sufficiently small with respect to the gradient vector. This occurs when the cost gradient is nearly perpendicular to the local constant-performance plane.

Next we present an example of this minimization algorithm applied to a simple problem. Suppose a channel's cost and an application's performance both vary in two parameters x and y. These could represent a channel's bandwidth and loss rate, for example. The channel's cost function is $c(x, y) = 10x^{0.1} + 5(1-y)^2$ and the application's performance function is $p(x, y) = x + 2(1-y)^{1.1}$. We negotiate such that $p(x, y)$ stays near 10. The application requests an initial flowspec of $(x, y) = (8.22, 0.1)$; the cost for this con-

figuration is 16.39. During iteration with variable step-sizes, the channel cost falls as follows:

iteration step 0	cost 16.39
iteration step 1	cost 16.14
iteration step 2	cost 15.86
...	
iteration step 20	cost 12.59
iteration step 21	cost 12.59

In 21 iterations the cost has fallen about 25%. The final flowspec parameters are (9.99, 0.98). If this same application used the same initial parameters with a different network that has cost function $c(x, y) = 0.3x^2 + (1-y)^{12}$, then 5 timesteps reduce the channel cost from 20.40 to 20.20. The final parameter point is (8.05, 0.022), quite different from the previous result.

This minimization algorithm may not reach a cost minimum in fewer steps than any other algorithm, but it does have some nice properties:

Assume the performance function $p(v)$ is differentiable. Then

$$p(v_0 + \Delta v) = p(v_0) + \frac{\partial p}{\partial v_1}\Delta v_1 + \ldots + \frac{\partial p}{\partial v_N}\Delta v_N + o(\|\Delta v\|)$$

When we project the cost gradient into the local constant-performance plane, we choose $\Delta v_1, \ldots, \Delta v_N$ such that

$$\frac{\partial p}{\partial v_1}\Delta v_1 + \ldots + \frac{\partial p}{\partial v_N}\Delta v_N = 0$$

Thus, at every step of the iteration, the change in the performance level is only $o(\|\Delta v\|)$.

Also, when iteration terminates (i.e. when the cost gradient is perpendicular to the constant-performance plane) the cost is a local minimum (or maximum) within the constant-performance plane.

> **Theorem:** When the cost gradient vector is perpendicular to the constant-performance plane at flowspec parameter point v_0, the cost function is a local minimum or maximum.

Assume: Assume that the cost gradient is perpendicular to the local constant-performance plane at v_0.

Proof: Linearize the cost function $c(.)$ at v_0.

$$c(v_0 + \Delta v) = c(v_0) + \frac{\partial c}{\partial v_1}\Delta v_1 + \ldots + \frac{\partial c}{\partial v_N}\Delta v_N + o(\|\Delta v\|)$$

By our assumption, the vector $(\frac{\partial c}{\partial v_1}, \ldots, \frac{\partial c}{\partial v_N})$ is perpendicular to the constant-performance plane. If a displacement d is within the constant-performance plane, then

$$c(v_0 + d) \approx c(v_0) + (\frac{\partial c}{\partial v_1}, \ldots, \frac{\partial c}{\partial v_N}) \cdot d \approx c(v_0)$$

Thus, when the cost gradient is perpendicular to the constant-performance plane, no small displacement within this plane results in a cost change.

One defect with this algorithm is that it only finds local constrained minima in the cost function. The iteration might become trapped in shallow local minima with much lower-cost channel parameters only a small displacement away. The algorithm could incorporate refinements similar to simulated annealing [114] to avoid this trap. The simulated annealing algorithm solves optimization problems by starting with a candidate solution and then considering perturbations of the candidate solution. The probability that the perturbation becomes the new candidate solution depends on the relative qualities of the two solutions, and the probability of accepting worse solutions decreases during the problem iteration. To apply this method to the minimization of channel cost, we could start with a candidate flowspec parameter point and then consider a move in any random direction away from the candidate. We make the probability of accepting the tentative move dependent on the relative costs of the two flowspec points, and as iteration continues we make the probability of accepting unfavorable moves smaller and smaller.

A problem with this optimization approach is that it may require orders of magnitude more iterations to reach a solution than a gradient descent method. For small numbers of

iterations, the algorithm discussed above probably performs better than the annealing algorithm for most performance and cost functions [114].

3.5.2.1 Negotiation Data Format

To support negotiations with this gradient descent algorithm, an application and network can exchange several different types of data. The application could send a description of its constant-performance plane at each iteration and could let the network calculate successive flowspec points. Alternatively, the network could transmit cost and cost gradient information to the application. Also, either the application or network could transmit a complete description of its performance or cost function to the other entity once at the start of negotiations. Then, the entire iterative procedure could be performed without any inter-agent communications. This would require a standard format for the description of cost or performance functions, however.

The performance of a communications application frequently depends on the interrelationship between characteristics of its different substreams. For example, an image transfer application could use two substreams to transmit high- and low-priority image data. The application could improve its performance by decreasing the cell loss rate on its low-priority substream. However, if the loss rate on the high-priority substream is relatively high, then the resulting image defects render the application performance independent of the loss rate on the low-priority stream. On the other hand, if a network's cost structure does not reflect possible resource-sharing between substreams, or if the amount of resource sharing between substreams is equal to the amount of sharing between independent channels, then the cost of a substream is independent of the characteristics of the other substreams in the same channel. In this case the cost of a channel is just the sum of the costs of its component substreams, each of which can be measured with the same cost function. In the simulation prototypes presented later in this paper, we make this simplifying assumption and have networks transmit their substream cost functions to their client applications. An application then can calculate its desired substream characteristics iteratively, without communication

with the network. A benefit of this approach is that networks with different control capabilities can describe their capabilities to applications at the same time that they transmit their cost functions. For example, a network that only can handle single leaky bucket loss bounds can specify this in its cost function.

3.5.2.2 Data Exchange Protocol

A system that implements the call setup negotiation procedure described above would have to contain a new protocol that could exchange the necessary channel description information. The design of such a protocol is outside of the scope of this report, but we mention briefly some recent works in the area of protocol specification.

Communications protocols typically are described using one of three paradigms. Finite state machine (or more generally, Petri net) descriptions give insight into transitions between protocol states, and for simple descriptions, analytical results can be derived about the protocol's correctness. Formal grammars have been designed for the specification of protocols. These grammars are simple enough that it is easy to verify the correct operation of a protocol designed with them; the LOTOS system is an example of such a language [119]. Unfortunately, it is difficult to design useful protocols using these limited languages. Recently, researchers have used more powerful computer languages for protocol specification. The additional power of these languages as compared to formal grammars facilitates the protocol design task, but complicates the analytic verification of protocols' correctness. However, computer tools have been developed that can verify a protocol's behavior exhaustively or with pruned-search tests [111]. These tools have been used to analyze the behavior of far more complicated protocols than was previously possible, and they have been used to find errors in formerly accepted correctness proofs.

Of course, these paradigms can be combined in the design of a complete protocol [105]. Petri nets can be used to model a protocol's control flow, computer languages can be used to model data structures, and formal grammars can be used to verify the correctness of parts of a protocol's behavior.

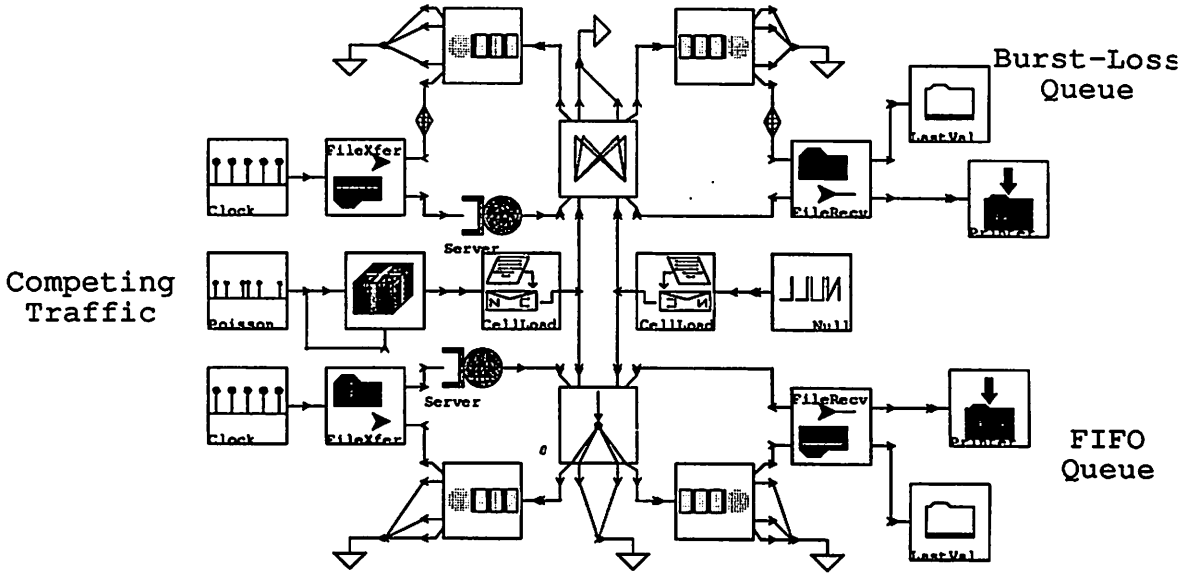
3.5.3 File-Transfer Application Prototype

This section presents a simulation prototype of a file-transfer application that can establish channels through a variety of networks using the Medley Interface negotiation method. The prototype is built within the *Ptolemy* system developed at U. C. Berkeley [106]. Ptolemy supports the integrated simulation of heterogeneous systems, for example systems that include synchronous signal processing subdomains as well as network components that are modeled in discrete-event domains.

The file-transfer source accepts packets of data from its high-level user and fragments the packets into smaller fixed-sized cells. The application uses a selective repeat protocol [38] to detect and recover from transmission losses. If any cell in a packet is lost, the file-transfer receiver does not acknowledge the packet and awaits its retransmission. All of the data sent by the application has equal importance to its users so it can be sent on a single substream. The throughput of the file-transfer application depends on the bandwidth and loss characteristics of its transport channel.

To prototype call setup negotiations that conform to the Medley Interface model we must calculate the performance function, in this case the packet throughput rate, for the file-transfer application. We must measure how the application's throughput varies with its sub-

stream rate and loss descriptions. The following experimental setup is used for these tests.



Ptolemy Diagram of a File-Transfer Application (Fig. 13)

The file-transfer application outputs a continuous stream of cells with no variation in the inter-cell spacing. Thus, a single leaky bucket with size 1 and rate R can describe the data rate exactly. The file-transfer data competes for buffer space with a Poisson source that models competing network traffic; this competition causes cell losses. Simulations with Markov-modulated Poisson competing traffic give similar results as long as the ratio of the traffic's peak rate to its average rate is sufficiently small.

We use two Ptolemy channel models. One uses a "flushing" queue (section 5.3) that can control the separation between cell losses, and the other channel uses a first-in, first-out (FIFO) queue that can not. Since either a single cell loss or the loss of several consecutive cells requires the retransmission of an entire packet, the file transfer application should benefit from control of its channel's loss burstiness.

The channels use loss bounds of the form LLB(TB(A, N), M, L). N is the size of a group of consecutive cell losses that is equivalent to a single cell loss. For this application N is the number of cells in a packet. M limits the allowed burstiness of losses of groups of cells, and L specifies the average rate at which losses of groups can occur. For a fixed L, M measures

how close $1/L$ is to the channel's group loss rate—the closer $1/L$ is to the loss rate, the larger is M. For each queue we fix L and parameterize the file-transfer throughput and the channel cost by R and M. We could have fixed M and varied L or varied both M and L, but here variations in M are sufficient to describe changes in the performance and cost functions.

Table 1 shows the ratios of packets received and acknowledged to packets transmitted (the *packet throughput ratio*) for the two channels as their queue sizes vary.

queue size	FIFO M: LLB(TB) max	FIFO channel throughput ratio	Flushing channel M: LLB(TB) max	Flushing channel throughput ratio
20	30	.77	18	.74
25	13	.80	7	.81
30	10	.81	5	.85
40	9	.85	5	.90
60	8	.88	4	.94

File-Transfer Throughput Performance (Table 1)

The FIFO channels use $L = 39$ and the flushing-channels use $L = 33$. The throughput rates do not depend very strongly on the LLB(TB()) maxima and are fairly proportional to the channel bandwidths. However, channel buffer requirements do vary significantly with the loss parameters.

The loss rates shown in table 1 might seem high to people used to working with high-quality communications applications. However, the file-transfer protocol is designed to give good packet throughput rates with quite lossy channels. This application thus can use very inexpensive channels with few resources allocated to them while still providing adequate performance to the end user.

For the FIFO channel, we can use a function linear in $(1/M)$ to approximate fairly accurately the falloff in throughput due to the nonzero cell loss rate. If we use least-squares

estimation with the data from table 1, the resulting throughput function is

$$throughput(M,R) = \frac{(1.11/M + 0.723)R}{cells/packet}$$

For the flushing channel we use another least-squares estimate to calculate the throughput function

$$throughput(M,R) = \frac{(1.02/M + 0.676)R}{cells/packet}$$

For the FIFO channel, violations of the TB() monitor better predict throughput than do cell losses. The number of packet losses per TB() violation varies from 0.6 to 1.2, but the number of packet losses per cell loss varies from 0.2 to 0.9, a larger range. This shows one benefit in using the LLB(TB()) monitor to describe cell loss characteristics rather than the average cell loss rate—applications can use QOS measures that are more closely correlated with their performances. TB() violations and cell losses both predict throughput about equally well for the flushing channel. This channel nearly always loses cells in packet-sized bursts, so the cell loss probability is nearly proportional to the cell loss *burst* probability.

We use data from the above simulations to design network cost functions also. The FIFO channel's cost depends on its bandwidth and rate allocation. The variation in M with queue size looks exponential, as might be expected for a queue fed by Poisson traffic. We make an exponential approximation and find the least-squares estimate for M as a function of queue size.

$$M = 29.7e^{-0.025 queuesize}$$

For the flushing channel, the exponential model does just about as well. Both models do

$$M = 11.56e^{-0.019 queuesize}$$

become less accurate for queue sizes outside of the range in table 1.

A network's cost function must balance a channel's bandwidth and buffer requirements. Thus, the FIFO channel cost function could be

$$cost(M,R) = \alpha(ln(29.7) - ln(M)) + \beta R$$

for some α and β. The flushing channel cost function could be

$$cost(M, R) = \alpha(\frac{0.025}{0.019})\ (ln(11.56) - ln(M)) + \beta R$$

The factor $(\frac{0.025}{0.019})$ ensures that both channels charge the same for identical resource consumption.

During call setup negotiations, the file-transfer application tries to find the lowest-cost channel for a fixed throughput rate. The application picks an initial flowspec parameter point that yields the desired throughput and then uses the iterative strategy discussed in section 3.5.1 to find the minimum-cost channel. The results of course depend on the α and β parameters of the network's cost function. These parameters differ in different networks because of differences in the relative costs to provide buffer space and transmission bandwidth. Because of these differences, the same file-transfer application would use different channel parameters with different networks.

Suppose a network with FIFO queues has a cost function with $\alpha = 20$ and $\beta = 1$. The file-transfer application chooses an initial bandwidth R value of 1000 and an initial M of 15. After iteration, the application obtains an R of 746 and an M of 3.0. During negotiations, the cost of the channel decreases by 22%.

A different network that also uses FIFO queues has a cost function with $\alpha = 45$ and $\beta = 1$. The file-transfer application chooses the same initial channel parameters. Iteration produces an R of 952 and an M of 9.6. During negotiations, the cost of the channel decreases by 2.8%. As expected because of the larger α value, the final parameters allow more losses than with the previous network.

With both networks, if we were to begin negotiations with a different channel parameter point that yields the same packet throughput rate, then we would end up with similar parameters after negotiation. For the first network, if we start negotiations with $R = 797$ and $M = 4$ then negotiations yield $R = 731$ and $M = 3.0$. Negotiations with the second network terminate immediately with $R = 797$ and $M = 4$.

If the same file-transfer application has access to a network that uses flushing queues, the greater efficiency of these queues can lead to a cost savings. Negotiations would proceed similarly to those with the FIFO queues, but the final costs should be lower to reflect the smaller resource needs. Table 1 shows that flushing queues give the same throughput ratio as FIFO queues with about 15% to 25% less buffer space. During periods of extreme network overload, flushing queues are even more advantageous (section 5.4).

3.5.4 Video Application Prototype

Next we describe the simulation of a compressed video transfer application, also done in Ptolemy. The video application uses conditional replenishment, a fairly simple compression technique, to reduce the data rate of a video stream before transmission. A conditional replenishment coder divides each frame of video into square blocks all of the same size. Both the transmitter and receiver store a copy of a "state" image that initially contains the first frame of the video sequence. Whenever a new input frame is received at the source, the source compares each block in the input with the corresponding block in the state image. If the input block is similar to the state block, the input is ignored. If the input block is sufficiently different (by a mean-square difference comparison criterion), then the input block is transmitted to the receiver and also is copied into the source state image. The receiver copies all blocks that it receives to its own state image and displays its updated state image at the frame rate.

The Ptolemy prototype adds a few enhancements to this basic algorithm. First, the transmitted blocks are coded and sent over two separate substreams. The three most significant bits of the update blocks are sent over a "high-priority" substream (with substream identifier *hipri*) and the next three significant bits are sent over a "low-priority" substream (with substream identifier *lopri*). Also, a problem with the generic conditional replenishment algorithm is that if a block is lost during transmission and no changes occur at that block position for a long time, then the receiver will display an error at that position for a long time. To reduce the persistence of errors, the prototype conditional replenishment cod-

er generates a third "correction" substream (with substream identifier *correction*) over which each block in the source state image is sent periodically. These blocks may experience high transmission delays. When the receiver reads a block from this substream, it compares the block's creation time with the creation time of the corresponding block in its state image. If the correction block is newer, it is copied into the state image.

The design of a performance function, a measure of video quality for this application, is more complicated than for the file transfer application because of the larger number of substreams it uses. There are many more substream characteristics to be negotiated. We simplify the performance function in several ways. First, the video application only provides an acceptable performance when some of the substream parameters fall within known ranges. Also, some of the substream parameters are constrained by others. For example, the delays of the *hipri* and *lopri* substreams must be equal and the rate descriptions of these substreams are equal as well, because of the structure of the compression algorithm. These constraints limit the domain over which the performance function must be defined.

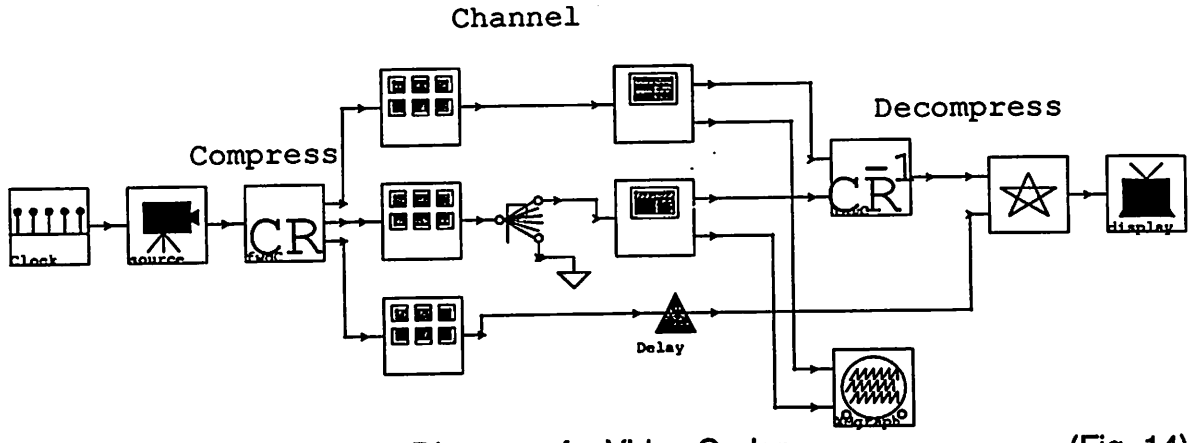
The output video quality of the application varies very sharply with the loss rate on the *hipri* substream. The effects of variations in this parameter dwarf the effects of variations in other substream characteristics. It is fair to say that for a fixed performance level, the loss rate on the *hipri* substream must be fixed.

For simplicity we assume that the loss rate of the *correction* stream is 0. Since this substream can tolerate delays much larger than the other two substreams, lossless transmission could be implemented with an acknowledge-repeat protocol as was used by the file-transfer application. This assumption simplifies the receiver since it need not detect errors on the correction substream.

The remaining channel parameters that can be varied are the data rate of the *hipri* substream, the data rate of the *correction* substream, the delay of the *correction* substream, and the loss rate of the *lopri* substream. With the Ptolemy simulator we have run simulations to compare the output video performance level that results with a wide variety of flowspecs.

Using these tests we can design this application's performance level-set.

Multipriority Conditional Replenishment



Ptolemy Diagram of a Video Coder (Fig. 14)

With four parameters to vary, the simulations cannot be organized as neatly as were the file-transfer simulations. We begin by finding the smallest compression ratio that justifies performing compression and the largest that gives an acceptable picture quality. A compression ratio of 4:1 justifies compression and gives an acceptable picture quality even at moderately high levels of data loss on the *lopri* substream. A compression ratio of 10:1 gives about the same picture quality at much lower loss levels. We only study *hipri* bandwidth levels between these compression limits.

We next study the trade-offs between the data rate of the *hipri* substream and the loss rate of the *lopri* substream. We measure date rate with an RLB(*hipri*, 2000, *K*) monitor; a burst-level of 2000 allows frames after scene changes, which are not compressed by conditional replenishment, to be transmitted without violating the rate monitor. We monitor losses with an LLB(*lopri*, 50, *L*) monitor. The bandwidth of the *correction* substream is fixed at 1/5 of the original data rate and the delay of this substream is set to 0. The following combinations of *hipri* data rate and *lopri* loss rate give fairly equivalent subjective quality

video.

hipri stream *K* (kilotokens per second)	lopri stream *L* value (deliveries per loss)
18.1	29
13.4	100
9.4	333
6.4	1000

Next we study the trade-off between the data rate of the *correction* stream and the loss rate of the *lopri* substream. The data rate on the *correction* substream is constant, so it can be described with an RLB(*correction*, 1, *K*) bound. We fix the *hipri* substream data rate with a *K* value of 18.1 kilotokens per second and fix the delay of the *correction* substream to be 0.

correction stream *K* (kilotokens per second)	lopri stream *L* value (deliveries per loss)
12.9	29
6.5	50
3.25	100

We continue studying pairwise trade-offs between various parameters to learn more about the nature of this video coder. Some intuition becomes clear. At higher rates of *lopri* cell loss, the video quality varies more sharply with the parameters of the *correction* substream than at lower rates. However, at low *hipri* bandwidths and low *lopri* loss rates, to decrease the *correction* delay or to increase the *correction* bandwidth helps somewhat in hiding compression artifacts. Further, delays longer than eight frame times are of little use except for very slowly changing video sequences.

With the intuition developed during the tests, we produced t application's performance level-set. We first chose a vector of baseline flowspec parameters that we judged to produce video with coding defects that are barely noticeable but certainly do not distract from the contents of the video scene. For the subjective tests, we chose several groups of three flowspec parameters and then varied the fourth parameter until subjective evaluators judged the resulting sequences to be of the same quality as the baseline sequence. The subjective evaluators were graduate students at U. C. Berkeley not studying video coding; they were not told in advance the video processing techniques employed in these tests or nature of the video impairments they were judging. The testers were shown two 45-frame video sequences that looped repeatedly—the baseline sequence and a test sequence. They were asked which they found to be of higher quality. If, without prompting, a tester said that the sequences' qualities were equal, then the test sequence parameters were entered into the performance level-set listed in table 2.

hipri stream K (kilotokens per second)	lopri stream L value (deliveries per loss)	correction stream K (kilotokens per second)	correction stream delay (frame times)
7.1	1000	12.9	1
7.1	1330	12.9	4
7.1	2000	12.9	8
7.1	2000	6.5	1
7.1	2500	6.5	4
7.1	3330	6.5	8
7.1	3330	3.2	1
7.1	4000	3.2	4
7.1	5000	3.2	6
9.4	333	12.9	1
9.4	455	12.9	4
9.4	667	12.9	8

hipri stream *K* (kilotokens per second)	lopri stream *L* value (deliveries per loss)	correction stream *K* (kilotokens per second)	correction stream delay (frame times)
9.4	667	6.5	1
9.4	833	6.5	4
9.4	1110	6.5	8
9.4	1110	3.2	1
9.4	1330	3.2	4
9.4	1670	3.2	8
13.4	100	12.9	1
13.4	220	12.9	4
13.4	333	12.9	8
13.4	250	6.5	1
13.4	333	6.5	4
13.4	500	6.5	8
13.4	333	3.2	1
13.4	500	3.2	4
13.4	1000	3.2	8
18.1	29	12.9	1
18.1	67	12.9	4
18.1	100	12.9	8
18.1	50	6.5	1
18.1	100	6.5	4
18.1	145	6.5	8
18.1	100	3.2	1
18.1	145	3.2	4
18.1	250	3.2	8

Channel Parameters that Yield Constant Performance (Table 2)

In the four-dimensional space of flowspec parameters, this data forms a three-dimensional

subsurface over which the video coder performance is nearly constant. This performance level-set can be used during call setup negotiations to establish minimum-cost channel parameters. We could attempt to estimate this entire surface with a multidimensional polynomial that would be used as a performance function during negotiation, but this approach has two drawbacks. Low-order estimates might differ significantly from the true surface. However, higher-order polynomial approximations would contain oscillations that would make the approximation useless for gradient estimation. Instead, we estimate local constant-performance planes near a point with linear approximations to the performance level-set.

The video application begins negotiations by picking a flowspec parameter point p that yields the desired video quality. To calculate the next point during channel-description iteration, the application must estimate the constant-performance plane near p. This can be done as follows. Assume that the application's channel is described by D parameters.

- Find D points from the performance level-set close to p. One way to do this is to choose points x_i from the performance level-set that have channel-description parameters that are just less than p's in all but the i^{th} coordinate; the i^{th} coordinate of x_i is just larger than that of p. Alternatively, we could define a metric function in parameter space and use the function to find the points from the performance level-set nearest p.
- These D points define a D-1 dimensional hyperplane in the space of flowspec parameters. The normal vector v to this hyperplane can be found by solving the equation

$$\begin{bmatrix} x_1 \\ \cdot \\ \cdot \\ \cdot \\ x_D \end{bmatrix} v = \begin{bmatrix} 1 \\ \cdot \\ \cdot \\ \cdot \\ 1 \end{bmatrix} \qquad \text{(EQ 1)}$$

where each of the x_i's is a vector of its component parameters.

> **Theorem:** The D points x_i define a D-1 dimensional hyperplane defined by equation 1.

Proof: A normal-form equation of this hyperplane is $(x - x_1) \cdot v = 0$. This is equivalent to $x \cdot v = x_1 \cdot v$. This equation must be satisfied for $x_2, \ldots, x_D$. Further, the value of $x_1 \cdot v$ may be set to any nonzero value by scaling the components of v. Choose this value to be 1 and equation 1 results.

- With this estimate for the constant-performance plane, the rest of the negotiation algorithm can proceed as specified previously.

During call setup negotiations, the conditional replenishment coder chooses points from its performance level-set near the current flowspec parameter point using both a metric function and individual coordinates. First the coder sorts all points in the performance level-set by their distance from the current point; the closest point is x_1. Then, points x_2, ..., x_D are chosen from the sorted list such that the i^{th} coefficient of the current point is between the i^{th} coefficients of x_1 and x_i. This method produces more accurate negotiation results than does choosing points x_i with a metric function alone. We have tried several different forms for the metric function, and all that give appreciable weight to all flowspec parameters produce similar negotiation results.

Some complications arise when using the performance level-set in table 2 for negotiations. First, because the *lopri* substream L values are much larger than the substream bandwidth and delay values, gradient and gradient projection calculations accumulate significant inaccuracies. The loss rate component of both the application performance gradient and the network cost gradient is two or three orders of magnitude larger than the other components. Whenever these gradient components have opposite signs, the cosine of the angle between the gradients is within a few parts per million of -1 so iteration terminates. (If both gradients have one component 10^3 times larger than the others, then the law of cosines implies that the cosine of the angle between them is within a few parts of 10^6 of 1 or -1.) Thus, channel cost reductions through trade-offs among the other flowspec components do not occur. To utilize these trade-offs we perform renormalization [109]. If we multiply the L values by 0.01 and adjust the network cost function accordingly, then common numerical accuracies suffice and negotiations treat significantly changes in all flowspec pa-

rameters.

It is possible that during the calculation of the constant-performance plane, the negotiation algorithm will detect an ill-conditioned problem. This occurs when the points from the performance level-set that are chosen to approximate the constant-performance plane near the current point do not define a proper N-1 dimensional subspace of the N dimensional parameter space. To remedy this problem, the negotiation algorithm adds another point from constant performance surface and finds the v such that $|Xv - \mathbf{1}|_2$ is minimum; this replaces equation 1. The simplest way to solve this minimization problem is with the normal equations $v = (X^tX)^{-1}X^t\mathbf{1}$. However, QR factorization gives much more accurate results for about only twice as much computation [109]. The QR factorization method factors X into an orthogonal matrix Q and an upper triangular matrix R and then solves the equation $Rv = Q^t\mathbf{1}$.

We perform some negotiations using the constant-performance plane described above with channel cost functions similar to those derived during the discussion of the file-transfer application. Of course the cost functions must be extended to handle a channel with three substreams rather than one and to charge appropriately for a substream with variable delay. We use cost functions of the form

$$cost = 2\alpha K(hipri)^{\gamma} + \beta(\log(L(lopri)) - \log(0.29)) + \alpha K(correction)^{\gamma} + \kappa Delay(correction)^{\mu}$$

The "log(0.29)" term comes from the assumption that no substream has an L value smaller than 29; 29 is renormalized to 0.29.

First we use a cost function with $\alpha = 0.2$, $\gamma = 0.5$, $\beta = 0.6$, $\kappa = 3.0$, and $\mu = -0.5$. We begin a negotiation with these flowspec parameters: *hipri* substream bandwidth monitor K = 9.42 kilotokens per second, *lopri* substream loss monitor L = 1330 deliveries per loss, *correction* substream bandwidth monitor K = 3.27 kilotokens per second, and *correction* substream delay = 4 frame times. The initial channel cost is 5.38. After 44 iterations the

flowspec parameters are (7.76, 1440, 1.89, 5.46) and the channel cost has fallen to 5.01. If we start with flowspec parameters (13.5, 500, 3.27, 4) then the channel has an initial cost of 5.04. After 56 iterations the flowspec parameters are (17.0, 341, 3.29, 8.0) and the channel cost is 4.55. A third negotiation starts with parameters (18.2, 1420, 2.58, 4) and channel cost 5.86. After 52 iterations the parameters are (16.6, 1250, 2.68, 8.0) and the channel cost is 5.28.

Next we use a different network for which low-loss substreams are less expensive. This network's cost function uses $\alpha = 0.2$, $\gamma = 0.5$, $\beta = 0.1$, $\kappa = 3.0$, and $\mu = -0.5$. We perform negotiations with the same three initial channel configurations as above.

Initial parameters	initial cost	iterations	final parameters	final cost
9.42, 1330, 3.27, 4	3.47	43	7.73, 1460, 1.89, 5.78	3.03
13.5, 500, 3.27, 4	3.62	54	14.7, 694, 4.04, 8.0	3.31
18.2, 1420, 2.58, 4	3.92	54	16.6, 1400, 2.72, 8.0	3.41

Although the parameters after the three negotiations are somewhat different, the channel costs are similar. As expected, final costs are lower with this network than with the first, and the final parameters use more *lopri* bandwidth.

Now the application uses a different network for which bandwidth is more expensive. This network's cost function uses $\alpha = 0.2$, $\gamma = 0.7$, $\beta = 0.1$, $\kappa = 3.0$, and $\mu = -0.5$.

Initial parameters	initial cost	iterations	final parameters	final cost
9.42, 1330, 3.27, 4	4.26	41	7.42, 1410, 1.89, 5.21	3.64
13.5, 500, 3.27, 4	4.72	52	11.4, 298, 1.88, 7.13	3.87
18.2, 1420, 2.58, 4	5.33	46	14.0, 1400, 3.06, 8	4.42

These negotiations reduce channel costs between 15% and 18%.

It is important during negotiations that enough points from the performance level-set

be sufficiently close to the current flowspec parameter point that the constant-performance plane can be estimated accurately. Further, if negotiations carry the current point past the boundary of the performance level-set then negotiations must be stopped. Presumably the application designer does not want the application to operate outside of the performance level-set's range. For example, several negotiations above terminate when the correction substream delay increases to 8.

The two photos below show frames that were transmitted with the initial and final flow-spec parameters of the final negotiation above.

Frame Transmitted over Channel with Initial Negotiation Parameters (Photo 1)

Frame Transmitted over Channel with Final Negotiation Parameters (Photo 2)

3.6 Medley Interface Implementation Issues

The Medley Interface channel setup model only addresses one component of broadband communications network design. This section discusses how the Medley Interface model interacts with other network components and protocols.

3.6.1 Interaction with Existing Protocol Hierarchies

It is important that the Medley Interface model have a minimal impact on existing network protocols if it is to be accepted and integrated seamlessly with current systems. BISDN proposals divide their functions into separate "planes." The User Plane is responsible for data transfer between network endpoints. The Control plane handles call control and network signaling. The Management plane provides network management and monitoring support, and coordinates the actions of the other two layers [104, p. 1-3]. The Medley Interface is used during call-setup to establish characteristics of transmission channels; it does not provide transport services itself. Thus, the Medley Interface could be integrated into the "Control Plane" of proposed BISDN networks with fairly minimal impact on other

planes.

Current BISDN proposals further divide their actions into "layers" orthogonal to the above planes. These layers are analogous to the layered structure of the OSI protocol stack model. While actions in separate planes provide either transport, connection, or management functions, actions in higher layers provide more abstract or high-level services than actions in lower layers. The BISDN Physical Layer is responsible for the transport of fixed-size data units between connected, specified endpoints. The BISDN ATM Layer is responsible for routing, generation of the fixed-sized data cells, etc. The BISDN ATM Adaptation Layer (AAL), discussed in section 2.7, provides application-specific functions such as timing recovery, segmentation, and error detection.

The Medley Interface model affects rate and QOS descriptions at the cell level only, and just uses services of the physical layer. It affects the call setup and control functions of the ATM (cell transport) Layer through that layer's Control Plane, but it does not affect the actions of the ATM Layer User Plane. Also, the Medley Interface model may replace functions of the ATM Adaptation Layer (section 2.7), such as lost-data retransmission.

3.6.2 Information Format in Cell Payload

Applications that obtain Medley Interface connections with more than one substream must specify the substream to which each transmitted cell belongs. The substream identifier could be placed in the beginning bits of each cell payload, just as the ATM Adaptation Layers embed application-specific information in cells. Medley Interface connections with only one substream would not need to embed any substream information. Connections with two substreams would use the first bit of each cell payload to specify the substream to which the cell belongs. Connections with more substreams would use more bits. Since the number of substreams allocated to a connection does not vary over the lifetime of the connection, there is no ambiguity about the number of bits in each cell that are used for the substream identifier.

This encoding method is simple and amply powerful for all reasonable applications.

With ATM cell sizes currently specified as $384 = 8 \times 48$ bits, up to 2^{384} substreams could be allocated to every network connection. Even $256 = 2^8$ substreams could be allocated to a single connection with an overhead penalty of only $8/384 \approx 2\%$.

3.6.3 Application and Network Interface Software

The Medley Interface model affects application designs in several ways. Most importantly, the model expects that applications know enough about their transport needs to negotiate them with the network. (All integrated-service networks have this requirement in some form.) With the Medley Interface model, applications know about the availability of specific transport services, and they can request specific characteristics for each substream. An advanced application thus can adapt its source coding technique if requested transport services are not available. Alternatively, an application could choose to support only a subset of its total capabilities if attached to a limited network. Applications that can utilize many types of networks through a common interface should be more widely used and more popular than network-specific, fixed-capability applications.

For example, a real-time videoconferencing application could choose among the coding techniques described in section 4.4 to obtain best performance with a range of flowspec parameters. If substreams with very low loss rates are available and sufficiently inexpensive, the videoconferencing application could use motion compensation with simple periodic replenishment for compression. If only high-loss and bursty-loss substreams are affordable, the videoconferencing coder could use conditional leaky motion compensation. This technique requires more computation than periodic replenishment, but it hides the effects of cell losses better.

If only low-bandwidth connections are available to a particular destination, the videoconferencing application could subsample the video at its source and interpolate between subsamples at the receiver. Although the subsampled video will not show the fine details of the original video sequence, for many users it is preferable to receive lower-quality images than no images at all.

3.6.4 Application Multiplexing and Coding Description

An application that can select between a variety of signal coding methods must tell its receiver about its chosen data multiplexing strategy. The source could signal that one of a set of previously agreed-upon methods has been chosen. Alternatively, at startup a source could send the destination a map from each substream ID to a particular data type. For some applications it might be simplest if each data cell contained enough information to tell how the cell should be interpreted.

3.6.5 Network Channel Provision

A network uses the information obtained from the Medley Interface to implement a channel with the requested characteristics. The network maps the requested substreams to one or more virtual circuits; just as many virtual circuits can share a single physical link, many substreams can share one virtual circuit. To obtain different qualities of service for different substreams sharing a single virtual circuit, the network call-setup processor can use the priority mechanism of virtual circuits and can alter the substreams' multiplexing and buffering pattern at the network interface. If it is significantly less complicated to send many substreams over a channel with better QOS than they need than to establish many virtual circuits for the substreams, the interface could send the streams over one channel, deliberately wasting network resources but reducing management complexity.

A network could use information obtained from the Medley Interface to do a better job of routing connections. Multiple virtual circuits supporting the same channel could be handled by different routes in order to better balance the traffic load or to use network switches with special features for one of the virtual circuits. If multiple virtual circuits supporting one channel have no interrelated delay bounds specified, lower-QOS virtual circuits could be routed over high-delay but lightly utilized routes, increasing network utilization and efficiency.

With the Medley Interface, a network can process different substreams with different buffer management disciplines. For example, lossless substreams would be given absolute

loss priority over low-priority substreams. Awater [35] shows how one buffer management discipline can simultaneously support a low-delay channel and a low-loss channel, although he does not propose the use of different buffer access and service disciplines for different channels' specific quality of service needs.

The Medley Interface could request AAL-level processing at the endpoints of a connection. The network could add error-correction coding or buffering to eliminate delay jitter, for example.

3.7 Benefits of the Medley Interface Model

This section reviews some of the benefits of the Medley Interface. First, the Medley Interface allows applications to describe their QOS requirements and rate parameters in more detail than is now allowed, enabling more efficient network resource allocation. With a standard BISDN interface, scarce or expensive resources such as bandwidth or buffer space must be allocated based upon only a few flowspec parameters. In practice, estimates of network traffic statistics are far from perfect. The rate of call requests is time-varying and difficult to predict. Further, application-supplied descriptions of their traffic statistics are incomplete and imperfect as well. For a network to establish a data connection with quality of service guarantees thus requires that the network make conservative assumptions [49] that allow the network to fulfill its QOS guarantees but leave network resources underused. The Medley Interface model's substream decomposition and detailed flowspec format help reduce resource waste by allowing networks to allocate resources less conservatively. Further, this model facilitates complex negotiations between applications and networks, allowing applications to find the most cost-efficient channel that supports a given performance level with a variety of networks.

A network interface must be clearly defined and well-known to the network's client applications. However, if the *implementation* of a transport channel is separated from its interface then network providers can upgrade network algorithms or components without making existing applications obsolete. If the interface is sufficiently general, new commu-

nications applications can use networks that implement the interface without changes to the network. This flexibility is one of the most important benefits of a useful interface model.

3.7.1 Specialized Buffer Management

The Medley Interface model allows network components to tailor their buffer management disciplines to the data on different substreams. The network could use the buffer management disciplines discussed in chapter 5 to implement substreams with specific loss and loss-burstiness characteristics. For example, given a suitably detailed description of its local traffic, each network node could construct an appropriate *priority screening* discipline that would allow all of the substreams that make up a videoconferencing channel to share the same buffer and output link. The design of this buffer management discipline gives each substream different qualities of service. Also, one of an application's substreams could be buffered so that any losses occur in very short bursts while another of its substreams is buffered so that the time between loss bursts is maximized.

With different buffer management disciplines applied to different substreams it is feasible for networks to offer lossless transmission as a viable transport service. Simpler cell-relay network interface models often implement lossless transmission channels with high-layer protocols that detect cell losses and request retransmission of the lost data. These *automatic repeat request* (ARQ) protocols still are supportable within the Medley Interface framework and are certainly appropriate for data streams that need lossless transmission and can tolerate high delay and delay jitter. However, the Medley Interface format can describe substreams that need both lossless transmission and stringent timing bounds. A network implements a lossless substream as if it were circuit-switched; the substream needs guaranteed buffer space and guaranteed maximum bounds on the service delay at every network node. As a result, buffers and service timeslots for lossless substreams cannot be shared with other data streams; further, these resources must be reserved in sufficient quantity for the worst-case traffic characteristics of the lossless substream. Obviously, to support such a substream is quite expensive and clearly is not suitable for most traffic. Lossless

substreams may be appropriate for small subsets of an application's traffic, however. For example, the video compression algorithm in a videoconferencing application may switch between several different quantizer tables in order to best match the quantizers to the current input data. The video coder must notify the receiver about the quantizer table change, and if that information were lost then all of the receiver's decoded video would be substantially in error. Data representing quantizer table changes would require only a very small bit-rate, but they certainly would benefit from lossless transmission. Chapter 4 contains additional examples of applications that benefit from sparing use of lossless Medley Interface substreams.

Either ARQ protocols or worst-case resource allocation allows a network to guarantee to an application that no transmitted cells will be lost. Some researchers have considered *forward error-correction* (FEC) methods for the reduction of cell loss rates [71]. However, this approach would introduce delay into the decoding of every cell because both the cell and its error-correction data would need to be received entirely before errors could be corrected within the cell. Also, an error-correction code long enough to correct for the loss of every single bit in a cell would be fairly complicated. Finally, the mechanism of cell loss is quite different from that which causes bit errors. Cell losses are caused by localized congestion within the network, and if a cell were lost then the congestion is likely to affect the cells containing the error correction information also.

While FEC at the cell level may be too inefficient for most purposes, FEC may be useful to reduce the incidence of *bit errors* on some noisy channels. Some demanding applications may require more stringent bounds on the network's bit error rate than the network usually offers. For these applications, the network could choose to apply bit error detection coupled with cell retransmission or forward error correction. For wired and optical networks, bit errors are most likely caused by unpredictable electrical noise within the network's amplifiers and switches. The network components are designed for a target bit error rate, but forward error correction or detection could allow a much lower rate of undetected

errors. Since bit errors occur randomly and fairly independently for these networks [112], the probability of enough bit errors occurring in a single cell to defeat the cell's error detection or correction mechanisms is much lower than the probability of bit error itself.

3.7.2 Multiple Loss Priority Levels

Loss priority is a resource! Not only should a communications application be able to specify the loss rates for multiple substreams, but it should be able to say that no data on a particular substream should be discarded before traffic on another. A Medley Interface network allows an application to specify the order in which its network should discard data in the event of congestion. The loss-eligible bit in proposed ATM standards allows this to be done to some extent, but a Medley Interface network gives more levels of control. A videoconferencing application might choose to send all of its audio data more reliably than any of its video data. The application could choose to transmit a stereo audio signal over two substreams: one carries the average of the left and right channels and the other carries the difference between the two. For good audio fidelity, both substreams should be transmitted with nearly equal fidelity. However in the event of network congestion, if cells containing the difference signal were discarded before cells containing the average signal then the received audio quality will be much better than if losses affected the two substreams equally. To support this capability a network would need at least three loss priority levels.

3.7.3 Best-Effort Channels

The term "best-effort" used to describe a communications channel usually implies that a network makes almost no effort to transport data on the channel. The network allocates no resources for and guarantees no QOS to best-effort traffic, although the network probably gives this traffic as good a QOS as it can without adversely affecting the quality of traffic with QOS guarantees.

Applications might want to use best-effort channels because they are quite cheap and because the QOS that these channels provide often is good enough for the applications'

needs. However, with the Medley Interface it is possible to request channels with extremely low but guaranteed QOS. For example, a file-transfer application could request a channel with a delay bound of 12 hours. Networks could implement these low-quality channels in almost the same way as current "best-effort" channels, and thus could charge the same cost. The Medley Interface also can improve the QOS of "best-effort" transport service. Improved traffic description allows networks to decide more intelligently if resources should be allocated to best-effort traffic in some nodes to help reduce congestion in others. Further, applications that can tolerate very lossy or high-delay channels but do need to know bounds on their channel characteristics can obtain the needed information from the Medley Interface.

3.8 Conclusion

The Medley Interface is a model for the call setup interactions between communications applications and broadband digital networks. This interface organizes data transport into substreams; applications request and configure as many substreams as they need to model their communications needs accurately. The Medley Interface also presents a substream flowspec format that is intended to be simultaneously simple, powerful, and extensible. Cost-minimizing channel setup negotiations are possible within the Medley Interface model because the flowspec format is not excessively complex. The substream decomposition, flowspec format, and negotiations combine to enable network provision of channels with QOS guarantees that still use resources efficiently. The accurate QOS description possible with substreams and the Medley Interface flowspec format allow channel resources to be tailored to different application data types needs' without waste. Also, the accurate description enables more savings during channel setup negotiations, because negotiations can trade between a wide variety of channel characteristics to find low-cost channel implementations.

Simulations of negotiations with a file-transfer application and a video transport application both achieved channel cost savings as high as 20%. Negotiations with the video ap-

plication were more complicated than with the file-transfer application because of the greater number of flowspec parameters negotiated and because the video performance was evaluated subjectively. These negotiations used a technique that finds a low-cost channel flowspec whose parameters fall within the set defined by subjective performance testing.

The Medley Interface proposal should not require major changes to existing network architectures. Cell transport and network management protocols should be unaffected; call control protocols may need extension to support the degree of transport control that the Medley Interface allows. Further, hardware that implements ATM layer interfaces may need to be modified so that it can recognize substream identifiers as well as virtual circuit and virtual path identifiers.

Plenty of open questions remain in the design of the broadband network interface, and a few are introduced below. Some involve the optimization of difficult design problems over large systems, but many are economic—they can not be answered through engineering alone. They ultimately will be answered in the marketplace.

3.8.1 Resource Allocation and Pricing

The allocation of link and buffer resources for multiple calls through a single switch is somewhat understood. [33, 56, 59]. However, work still needs to be done to understand how networks should combine their resource allocation strategies, buffer management, and routing algorithms given statistics on the number and type of connections the network carries. This statistical control problem is complicated by the fact that although pricing is one method the network can use to control application behavior, the effects of different pricing policies on application behavior is difficult to predict.

A network's pricing policy charges for the use of resources that can not be used by others, and it helps compel certain behavior from applications. For example, a network may charge based on bandwidth usage, call setup costs, etc. It probably is infeasible to charge some amount for each cell transmitted. More likely, costs are determined by the agreed-upon traffic description, QOS specification, adherence to the traffic description, and actual

delivered QOS. In fact, it is unfair to charge based simply upon the number of delivered cells because this charges high-rate connections excessively for call setup and network management costs. However, the price of a given transport resource also is affected by the supply of the resource from other networks and by the demand for the resource from applications, both of which are difficult to model or predict.

3.8.2 Multicast Connections in Medley Interface Networks

A *multicast connection* connects possibly more than one source to possibly more than one receiver. Of course, several sources and receivers may be co-located, as with a multimedia terminal or a multiway videoconference with several participants at each site. The establishment of multicast connections in future broadband networks is very much an open research issue. Efficient algorithms are needed for routing multicast connections, adding and dropping individual connections from an existing multicast connection, etc. The specification of quality of service measures for a multicast connection also is quite difficult; if a data cell is delivered successfully to all intended recipients but one, should the loss be considered to affect only the one user's received quality of service or the QOS for the entire connection?

It is likely that not all participants in a multicast connection would want the same quality of service. Users could decide to receive only some of the substreams in a multicast connection and could specify different QOS parameters for the same substreams. The question of how to bill multiple users for partially shared transport service also needs study.

Chapter 4

FLEXIBILITY IN MODERN VIDEO CODERS

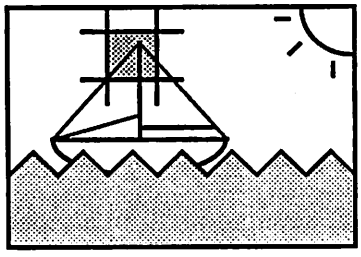
The previous chapter showed cell-relay networks and applications such as conditional replenishment video coders negotiating to obtain low-cost channels that support a fixed application performance level. This chapter discusses how more modern video coders can adapt to provide constant perceived image quality with the range of channel QOS specifications that might result from negotiations. Sections 4.4.3.3 and 4.4.3.4 present new versions of the traditional video motion compensation algorithm, developed by the author, that provide more resilience to cell losses than the original. These methods allow a video coder to operate with a range of channel bit-rates and loss rates. Section 4.5 presents negotiations with a modern video coder that varies not only its rate of channel losses but also the spacing between lost cells. During negotiations the video coder takes advantage of the subjective improvements possible with network control of cell loss burst lengths.

Video signals generally are compressed before transmission because the resulting reduction in transmission cost more than compensates for the compression effort. While video coding may not reduce a channel's loss rate, it can provide resilience to the visual effects of transmission losses by reducing the perceived objectionability of the resulting defects.

Video compression first removes redundancy from the source data. For example, picture regions that are relatively uniform can be represented more efficiently than by listing all of their pixel values. Also, compression suppresses information that is not perceivable by the human eye. Very high spatial frequency picture details or very small changes in brightness between nearby pixels can be ignored, reducing a video sequence's bit-rate but not changing its perceived appearance.

4.1 Discrete Cosine Transform

A widely used method for the compression of images is the discrete cosine transform (DCT) [23, 27]. The DCT is an example of a multiresolution method, in which an image is decomposed into several "frequency" components with a linear transform.



DCT Coefficients Result from a Linear Transform (Fig. 15)

The two-dimensional DCT divides an image into square blocks and multiplies the blocks on the left and right by orthogonal matrices (figure 15). Each of the two matrix multiplications performs a one-dimensional DCT on the input picture data—one on the data rows and one on the columns. After both one-dimensional transforms, the resulting coefficients are quantized and transmitted. The DCT removes redundancy by expressing largely uniform image regions with only one or two nonzero DCT coefficients. It suppresses perceptually insignificant information by more coarsely quantizing higher-frequency coefficients, to which people are less sensitive [28, 29]. For many video scenes and for DCT transform block sizes of 8 x 8 pixels or larger, most high-frequency DCT coefficients can be set to 0 without affecting the perceived quality of the reconstructed scene noticeably.

Moving video can be thought of as a three-dimensional signal, a brightness (or brightness-and-color) function of width, height, and time. We have seen that the two-dimensional DCT compresses images—the three-dimensional DCT could be used to compress video. The three-dimensional DCT transforms each of the rows, columns, and temporal slices of a video sequence with the one-dimensional DCT. The magnitudes of variations of different "frequencies" in each direction are represented by unique coefficients in the resulting three-dimensional DCT coefficient cube. If the sequence contains predominantly low-frequency variations, then most of the information about the sequence is contained in just a few coef-

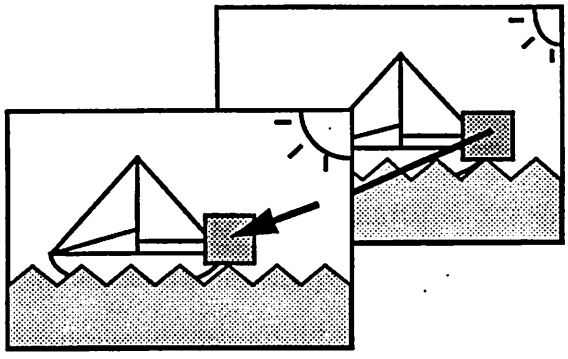
ficients of the DCT coefficient cube. (Most coefficients would be nearly equal to zero.) For compression, the coefficients near zero could be set to zero and then the resulting cube could be represented efficiently with entropy coding.

Unfortunately, video scenes with even moderate amounts of motion produce three-dimensional transform coefficients that often are not close to zero [14]. The nature of temporal redundancy in video differs from the redundancy found in two-dimensional images. In images, large regions are quite uniform, other regions vary gradually, and at the border between two regions there is an abrupt change. If we look at a single pixel in a video signal straight along its time axis, we see rapid changes that occur whenever a moving object covers or uncovers that pixel. To notice the redundancy present in video along its temporal axis, we must look from a pixel in the current frame to *nearby* pixels in neighboring frames. Such a compression method could eliminate temporal redundancy even for moving objects.

4.2 Motion Compensation

If we could compensate for the motion in a video sequence, then techniques such as the DCT would more successfully remove temporal redundancy from a scene. A technique called *motion compensation* does just that [30]. Motion compensation methods try to find regions in past video frames that are as similar as possible to some region in the current frame. "Pel-recursive" methods find a best-matching pixel in the previous frame for each pixel in the current frame. Region-based motion compensation methods segment each frame into "regions" that correspond to identifiable objects within the image, and then they try to track the motion of each region in the image. Most common are block-matching motion compensation methods that divide each frame into rectangular blocks and then search previous frames for a translated block of the same size that matches the current block as well as possible (figure 16).

The criterion used to decide when two blocks match usually minimizes the mean squared difference between a current block and all nearby blocks in the previous frame. Absolute differences and other criteria have been investigated as well [26].



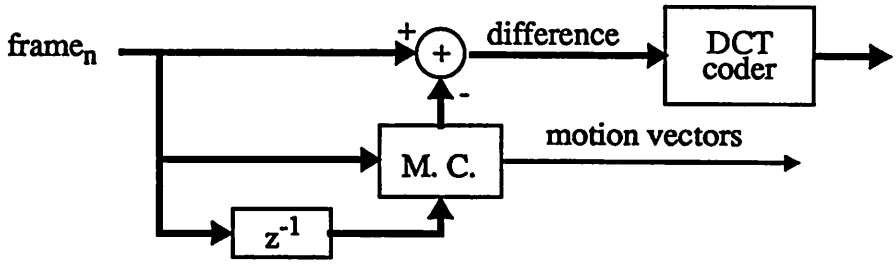
Block in Current Frame Estimated with Past-Frame Block (Fig. 16)

Once we have determined the trajectory of a block over several frames we could transmit the motion information, for example as a sequence of frame-to-frame displacement vectors. Then, we could perform DCT encoding, subband coding, or some other one-dimensional compression technique along the trajectory with much more effectiveness than if we had not compensated for the motion. Unfortunately, if we track the motion of a block for several frames from frame N to frame M, then because of non-translational motion and imperfect motion estimation, there will be parts of the frames between N and M that are not coded at all. These residual regions would have to be described and coded. Because of this problem, we know of no existing research that has used motion-compensated compression over more than two frames at once.

With groups of two frames, it is fairly straightforward to divide each frame into blocks and then to find the best-matching block in the other frame via motion estimation. Then, a short two-tap subband filter can provide some compression when applied along two-frame-long groups of motion-compensated blocks [14]. More common is to use differential encoding in combination with motion compensation. Rather than transmitting frames $f(n)$ directly, we transmit the first frame of a sequence $f(0)$ and then transmit the difference $d(n)$ between the current frame and the previous frame: $d(n) = f(n) - f(n-1)$. To decode this sequence, the receiver simply adds all received values: $\hat{f}(n) = \sum_{k=0}^{n} d(k)$. If changes from frame to frame are relatively small, then the number of bits needed to transmit $d(n)$ should be less than the number to transmit $f(n)$. Motion compensation with differential encoding

combined with the DCT usually provides two to three times more compression than the DCT alone at similar image qualities.

A typical motion compensation based video coder segments each input image into blocks. For each block, once the past-frame matching block has been found, the coder outputs the displacement vector that moves the past-frame block onto the current block. Also, the coder calculates and outputs the residual difference between the past-frame block and the current block. This difference may be nonzero due to non-translational motion within the video, brightness changes within the video, or motion displacement that is not a multiple of the motion estimation distance resolution. This difference is compressed with the



Motion Compensation Coder (Fig. 17)

DCT and transmitted.

To generate a frame of video, a motion compensation decoder reads in a motion vector and difference block for each block in the frame. The DCT encoding first is inverted. Using the motion vector, the decoder finds the block in the previous frame that the encoder used to calculate the difference values. By adding the received difference values to the past-frame block, the pixel values for the current block are produced. Note that the decoder needs a correct copy of the past video frame in order to generate a current frame. An intraframe coded frame of video can be sent from the encoder to the receiver in order to start the motion compensation process.

Motion compensation works well because motion vectors are a very efficient representation of motion, the most common temporal change in video. If most temporal changes in video scenes were brightness changes, scene changes, or other transformations, then mo-

tion compensation would not perform as successfully. Further, the difference signal produced by motion compensation coders is quite amenable to further compression by the DCT.

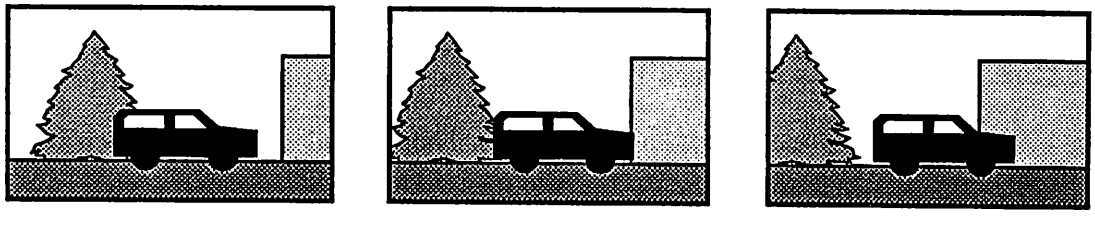
4.3 Relevant Standardization Efforts

The Joint Photographic Experts' Group (JPEG) of the International Organization for Standardization (ISO) produced what is commonly known as the JPEG standard to compress and represent continuous-tone (as opposed to two-level) images [97, 103]. JPEG is not intended for video compression, but since video is simply a sequence of frames JPEG has been used for video compression even though it does no interframe coding.

JPEG is based upon the DCT. Blocks of 8×8 luminance (brightness) values are discrete cosine transformed, quantized, run-length encoded, and entropy coded via Huffman coding or arithmetic coding. Chrominance (color) values are subsampled by a factor of two vertically and horizontally and then are also DCT-encoded, quantized, run-length encoded, and entropy coded.

The Motion Picture Experts' Group of the ISO designed the MPEG standard specifically for video compression [99]. MPEG uses a complicated version of motion compensation in which frames may be intraframe coded, differentially encoded from a previous frame (not necessarily the immediately preceding frame), or differentially encoded from both a past and future frame.

The combination of both past frame prediction and future frame interpolation helps alleviate the "uncovered area" problem. As the scene in figure 18 progresses from right to



Uncovered Region of Tree Caused by Automobile Motion (Fig. 18)

left, the car uncovers some regions within the frame and covers other areas. When coding

the center (current) frame, the newly uncovered lower right corner of the tree can be interpolated from the (future) frame on the right, and the lower left hand corner of the building can be interpolated from the (past) frame on the left. If either the past or future image were not available, then part of the current image would have to be coded without any reference block. In scenes with several moving objects, bidirectional interpolation allows many more blocks to be coded from good reference blocks than would one-direction predictive coding.

Since MPEG codes some blocks using blocks several frames in the future or past, the MPEG motion detection algorithm must be able to detect motion over a much larger region than traditional algorithms. Most likely, MPEG coders will estimate motion hierarchically. First, the coder estimates motion coarsely (to an accuracy of several pixels) without an exhaustive search but over a large area. Then, the coarse motion estimate is refined with a more accurate search centered at the coarse motion estimate. The coder can estimate motion to sub-pixel accuracy by interpolating values between pixel positions.

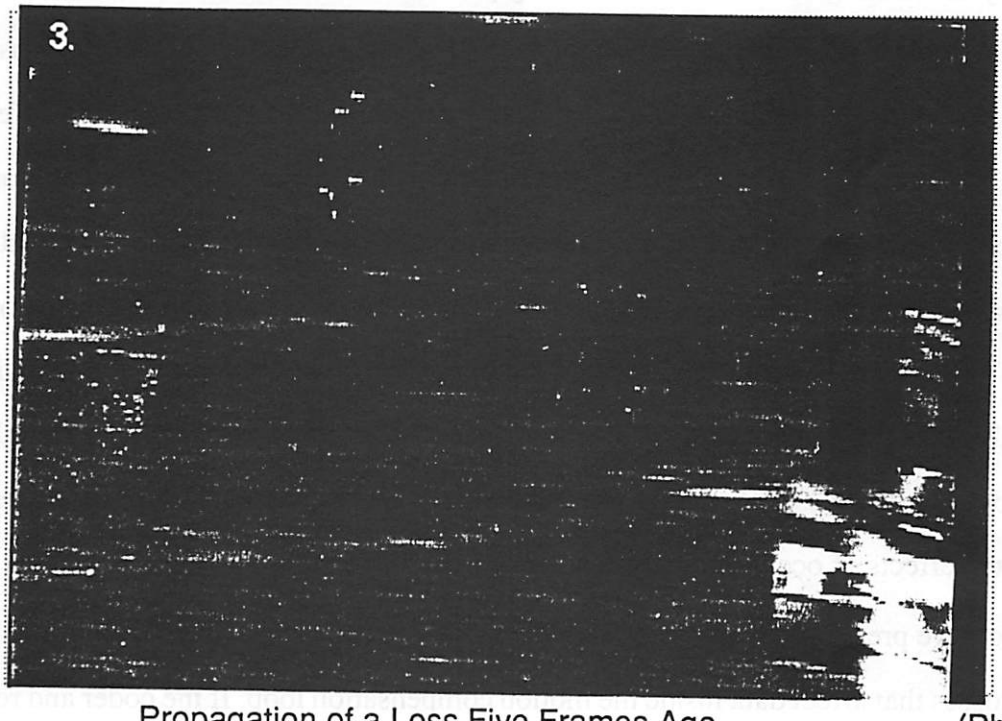
Other recent video standardization efforts include MPEG-2, which addresses higher-resolution video than the MPEG standard, and MPEG-4, which addresses lower-rate video. The H.261 standard of the International Telephone and Telegraph Consultative Committee (CCITT) specifies how to compress videoconferencing signals at multiples of 64 kilobits per second [100].

Two multimedia standards now being developed are the American National Standards Institute (ANSI) and ISO "HyTime" standard and the ISO "Multimedia and Hypermedia Information Coding Experts Group" (MHEG) standard [101]. These describe how applications can create and present information streams that combine video, audio, text, and graphics. Streams of the various types can be synchronized to each other or linked to viewer requests (such as button presses or menu selections). These standards do not specify signal coding formats themselves, but allow the formats to be specified as part of a data stream. For example, a multimedia stream could contain video coded with both the MPEG and H.261 standards and still images coded with the JPEG standard.

4.4 Video Coding for Lossy Networks

Chapter 5 presents network buffer management disciplines that implement channels tuned to the cell loss needs of a variety of communications applications. This section takes the opposite approach and studies ways that video coders based on motion compensation and the DCT can adapt be more resistant to the effects of cell losses.

If either motion vectors or the frame difference data are lost during transmission, then the video receiver will produce a somewhat incorrect version of the current frame. Since each frame is used at the receiver to generate the succeeding frame, unless corrective action is taken the effects of the loss will propagate to a larger area and become more easily noticed (photo 3). The problems of the corrupted current frame and of error propagation into



Propagation of a Loss Five Frames Ago (Photo 3)

future frames may be thought of somewhat independently. Estimation of lost data at the video receiver helps hide the effects of errors in the frame in which they occur. However, the motion compensation algorithm itself must be modified somewhat to eliminate the accumulation of even small errors. Two existing modifications are presented in sections 4.4.3.1 and 4.4.3.2; sections 4.4.3.3 and 4.4.3.4 present methods developed by the author.

4.4.1 Past Works

Most past research studying video transmission over lossy networks does not consider motion compensation [1, 3, 5, 6, 8, 9, 13, 19, 22]. Many of these works simply suggest the combination of layered coding, in which video is separated into more and less loss-sensitive streams, and multi-priority transmission, in which more sensitive data are transmitted at lower loss rates. [5, 6] discuss an alternative to the DCT that helps hide cell loss defects. [19] shows how filtering around the boundary of loss-affected picture regions helps to hide the severity of the errors.

A motion compensation coder can limit error propagation to a subset of DCT coefficients by calculating block differences using past-frame data that are the inverse-DCT of only the subset of coefficients with all other coefficients are set to 0. One previous study [18] uses only high-frequency DCT coefficients in the motion compensation difference. These authors argue that low-frequency coefficients cannot be coded with motion compensation because error propagation in the low-frequency coefficients makes received video quality unacceptable. Others [4, 16] argue that only low-frequency coefficients should be used in the difference operation. Any error propagation makes received video quality unacceptable, but low-frequency coefficients, packed in high-priority cells, essentially are never lost. In either case, other methods of eliminating error propagation must be used also or else the effects of occasional errors will accumulate enough to be objectionable.

Below we present and analyze several methods for using motion compensation in spite of cell losses that affect data inside the motion compensation loop. If the coder and receiver are designed correctly, high-quality video can be transmitted over lossy networks without abandoning motion compensation altogether.

4.4.2 Estimation of Lost Data at the Receiver

The loss of either the motion compensation difference signal or of motion vectors produces picture errors and error-propagation. If the positions of losses are known to a video receiver though, it can attempt to estimate or reconstruct the lost data, improving the quality

of the displayed video. Next, we discuss the recovery of lost motion vectors.

4.4.2.1 Recovery of Lost Motion Vectors

Both the horizontal and vertical components of adjacent motion vectors have intraframe adjacent-vector correlations that range between 0.1 and 0.3 for four tested sequences that contain varying amounts of detail and motion. The frame-to-frame correlation for the horizontal and vertical components of neighboring motion vectors is much higher in scenes with little motion but is smaller than 0.05 in sequences with rapid motion. Scenes with little motion can be reconstructed easily in spite of motion vector loss, so we focus on high-motion scenes. The above correlations indicate that both intraframe and interframe strategies for estimating lost motion vectors are reasonable.

The replacement of lost motion vectors with the component-by-component median of their intraframe neighbors seems to yield better image quality than simply the average of their intraframe neighbors. One explanation is that near the boundary between two objects in a scene, motion vectors for the different objects point in different directions. When a motion vector is lost, it should be replaced with a motion vector that describes the object from which the lost motion vector came. By taking the component-by-component median of the four nearest neighboring motion vectors, we less likely use a replacement value that averages in contributions from vectors that describe other displayed objects.

We have studied two recovery methods that use past motion vector values to reconstruct lost motion vectors. The simplest technique replaces lost motion vectors with the corresponding-position motion vectors from the previous frame. This actually works quite well—better than the intraframe median for some lost motion vectors and worse for others.

A more complicated recovery method searches all motion vectors in the nearby blocks of the frame before a lost motion vector. The past-frame motion vector that best moves its block into the position of the loss-affected block is used to replace the lost motion vector. Intuitively, if a past block moved from its old position into the loss-affected position, then that block probably continues to move along the same trajectory in the current frame. This

strategy actually does not work as well as the intraframe median method or past vector method. It occasionally is fooled along object boundaries, which causes boundary blocks to be reproduced incorrectly. These blocks stand out as too-bright or too-dark discontinuities along the objects' edges. However, any of these recovery methods produces much better-looking sequences than the replacement of lost motion vectors with a zero vector. Many fewer blocks stand out because of discontinuities with their neighbors.

4.4.2.2 Recovery of Lost DCT Coefficients

The estimation of lost higher-frequency DCT coefficients is not of critical importance. If the lost coefficients are set to zero then the loss-affected block is replaced by motion-compensated data from the previous frame; this works fairly well. If the receiver blurs the boundary (with low-pass filtering) of the loss-affected block, then the error is somewhat less noticeable [19].

The lowest-frequency DCT coefficient in each block is called the "brightness coefficient" or "DC coefficient" because it describes the brightness of its block. The loss of even a few brightness coefficients produces very objectionable image artifacts. People are very sensitive to incorrect brightness values in images, much more so than to incorrect detail. We can replace lost brightness coefficients with the average of their intraframe neighbors, the median of their neighbors, the average of intraframe and past-frame neighbors, etc. For these methods to work, lost brightness coefficients must not be adjacent.

We have found all of these methods to work fairly well—much better than no estimation at all. Blocks with estimated DC coefficients occasionally appear slightly too bright or too dark for their surroundings, but blocks with lost and unestimated DC coefficients stand out as sharp dark squares.

4.4.3 Motion Compensation Resynchronization

Estimation of lost video data helps reduce the magnitude and objectionability of loss artifacts. However, motion compensation causes even small errors to accumulate until they become annoying. Some resynchronization method beyond lost-data estimation is neces-

sary to prevent this error propagation.

4.4.3.1 Periodic Replenishment

The simplest resynchronization technique in motion compensation coders is to transmit periodically a frame of intraframe-coded video rather than of motion-compensated difference. The accumulated effects of all transmission losses before this synchronization frame are eliminated. This technique, *periodic replenishment*, produces an output with a bursty bit-rate however, since the intra-frame coded frames generally require many more bits than the difference-signal frames. Still, this method is used by the MPEG and H.261 video coding standards. A more sensible approach is to intraframe code a subset of every frame in such a way that all parts of a frame are intraframe-coded periodically. As the percentage of each frame that is intraframe coded increases error artifacts are eliminated more quickly, but the transmitted bit-rate increases. If B is the number of bits required to represent an intraframe coded frame, b is the number of bits required to represent a motion-compensated frame, and p is the percentage of data that is replenished every frame, then the number of bits required to transmit a periodic replenishment frame is

$$pb + (1\text{-}p)B.$$

On average, every pixel block is replenished every $1/p$ frames. If the probability q that a block suffers a transmission loss is much less than p, then a block is replenished many times between errors. The replenishments impose a bit-rate penalty but otherwise are harmless. If q is comparable to or greater than p, then several errors can increasingly corrupt a block before it is replenished.

4.4.3.2 Leaky Motion Compensation

Another approach to limit error propagation mimics leaky differential pulse code modulation (DPCM) coders [17, 24]. Rather than sending motion vectors and the difference signal

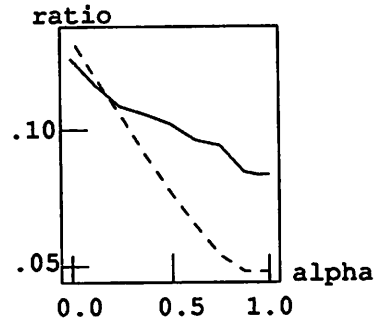
$$diff_k = input_k - input_{k-1}[motion\ compensated],$$

the coder sends motion vectors and a difference signal

$$diff_k = input_k - \alpha\, input_{k-1}[motion\ compensated].$$

The decoder adds α times its past frame to $diff_k$ to produce a new current frame. With α between 0 and1, each $input_{k-j}$'s contribution to $input_k$ is scaled by α^j. Thus, the effects of past errors decay away exponentially. Errors visually appear to fade away at a rate that depends on α. Subjectively, this error fading is preferable to the behavior of periodic replenishment, which causes errors to flash on the screen when they occur and flash again when they are corrected. Leaky motion compensation first was presented in [15] and later independently by the author of this report in [7].

As with periodic replenishment, leaky motion compensation trades between good error resilience and low bit-rate. As α approaches 0, errors decay away faster but more bits are needed to code the difference signal (figure 19). Images coded with $\alpha = 0$ have a compression ratio less than 1.0 because the DCT alone gives substantial compression.



Coded Bit-rate / Uncoded Bit-rate for Two Sequences (Fig. 19)

Since errors decay with this method but never disappear completely, we define the *half-life* of an error as the number of frames that must pass until the errored data is scaled by 1/2 in the current frame.

$$half\text{-}life = -1 / \log_2(\alpha)$$

4.4.3.3 Conditional Replenishment

Both of the above motion compensation resynchronization methods ignore the content of the coded images when deciding how frequently to resynchronize. The methods presented in this and the next section have been developed by the author to use the picture data when deciding whether to replenish a block; these methods have been presented in [7].

Some networks tell the receiver what image blocks have been lost; the receiver can reconstruct them partially using neighboring data in the same and past frames. The coder can aid the receiver by most frequently replenishing blocks that would be reconstructed poorly. For example, if the receiver replaces loss-affected blocks with the corresponding block from the previous frame then the coder need not replenish blocks that have not changed recently. Blocks that change most rapidly must be replenished most often.

Ideally, the subjective visual appearance of a block could be used to decide how often it is replenished. In practice, it is difficult to measure a block's subjective importance so other criteria are used. The source could measure the energy of the motion-compensated difference signal for a block—if it is higher than some threshold, the block contains an object undergoing non-translational motion or undergoing a color or luminance shift. Presumably, such a block represents a detailed foreground region in the video, so it should be replenished by transmitting it without interframe differencing. With this replenishment criterion, both the bit-rate and average error lifetime depend upon the threshold value and actual input video. The threshold must be matched with the network loss rate to best trade between the bit-rate and error lifetime.

This mean-square-error replenishment criterion clearly is imperfect, but it works quite well in practice. Compared to periodic replenishment, video coded with this method shows fewer and shorter-lasting errors.

The most annoying defect in video coded with this method is that some errored blocks stand out because their brightness is incorrect. A possible remedy is a replenishment criterion that checks the brightness change in a block from its previous value. If it is larger than some threshold then the block is replenished. Otherwise, the block is difference-coded. This criterion will not replenish a block if its brightness does not change with time. However, errors can affect such a block enough so that the errors are easily noticed at the video receiver. This brightness criterion alone does not produce high-quality video. It may be combined with a periodic criterion or another image-dependent criterion to yield better re-

sults.

Both of the above replenishment criteria decide whether to replenish a block based upon the block's difference from its immediate predecessor. Gradual changes in a block over time are not noticed. An improvement is to compare a block with the version of the block the last time it was replenished. Then, gradual changes cause a replenishment after the changes have reached a sufficient magnitude. Whenever a block is replenished it is saved to a "state" image for later comparisons. A coder that uses this conditional replenishment criterion does hide errors in slowly changing picture areas better than a coder that uses a frame-to-frame replenishment criterion.

4.4.3.4 Conditional Leaky Motion Compensation

We can combine the benefits of conditional replenishment and leaky-difference motion compensation by choosing different α values for each block; blocks with α values close to 0 are resynchronized more quickly than blocks coded with α values close to 1. Of course each block's α value must be transmitted to the receiver. This method can allocate more bits to the most critical portions of a video sequence and also can utilize external information such as network state.

As usual, a video coder must trade between loss resilience and bit-rate. Figure 19 shows that blocks with α close to 0 require more bits than blocks with α close to 1. (However, the choice of α values between 0.5 and 1.0 does not have a great affect on the bit-rate of the transmitted signal.) Conditional leaky motion compensation makes the trade-off in a very intelligent way. For example, in a stationary scene, background blocks can be predicted very well with past-frame blocks. So, if data for a background block is lost, the receiver still can produce an excellent picture. These background blocks can be coded with $\alpha = 1$ for maximum compression. Foreground blocks that change rapidly are more sensitive to data loss and should be coded with α closer to 0.

The same criteria can be used to choose α values as were used by the conditional replenishment method to decide which blocks should be intraframe coded. A very simple cri-

terion checks if the squared difference between a block and its motion-compensated previous block exceeds a threshold. If so then $\alpha = \alpha_0$; otherwise $\alpha = \alpha_1$.

Conditional replenishment methods that depend upon the changes in a block from frame to frame are quite simple. More complicated schemes that consider changes in a block over several frames, that use network congestion information to choose α values, or that utilize more than two α values could give better results.

If high-priority information such as a block's α value is lost, then the receiver probably replaces the loss-affected area with the block from the previous frame. If the block's motion vectors are available, the receiver can motion-compensate a block from the previous frame and use that to fill in the loss-affected area.

Of course, a block's α should depend upon the bit-rates needed to represent the differenced and undifferenced frames. If a block's pixel values can be coded with fewer bits or even only a few more bits than the frame-difference values then the transmitter should use $\alpha = 0$, winning both lower bit-rate and better loss-immunity. Undifferenced pixel values are most likely to take fewer bits than difference values during scene changes. However, it is difficult to predict the number of bits required to represent differenced blocks, undifferenced blocks, and blocks coded with numerous α values, without performing differencing with each α value and then coding each result. This would be quite expensive, since it requires that several DCT's, run-length encodes, and entropy codes be performed. It might be feasible to code just the differenced and undifferenced blocks and then interpolate the number of bits needed for other α values.

4.5 Medley Interface Negotiations with a DCT + Motion Compensation Based Video Coder

In this section we simulate and describe call setup negotiations between a video coder that uses motion compensation and the DCT and a Medley Interface network. Although statistical analyses give insight into the asymptotic behavior of network systems, real-world video coders do not produce easily quantified data rate distributions and actual networks do

not contain near-infinite buffer sizes and Poisson traffic. In some cases, cell losses depend very sensitively upon source rate characteristics, and artificially modeled video does not give useful results. For example, scene changes or frames with rapid motion require more cells than less active frames. (The peak-to-average ratio for the number of cells per frame of compressed video is about 4.0 for several different types of coders [87, 89, 90].) Further, coders are most sensitive to loss during the transmission of fast-moving detailed images, also when networks are most likely to lose data. To see how transmitted video sequences look, we need realistic cell loss patterns.

The simulations in this paper are performed with the *Ptolemy* heterogeneous simulation environment developed at U. C. Berkeley [106]. Ptolemy simulations are designed graphically—the Ptolemy user interface supports the design of hierarchical, block-based systems. Blocks included with the system include arithmetic and filtering functions, queues and switches, logic operations, signal sources and sinks, etc. Base-level blocks are written in C++, and new base-level blocks can be written and linked in by anyone familiar with C++ programming.

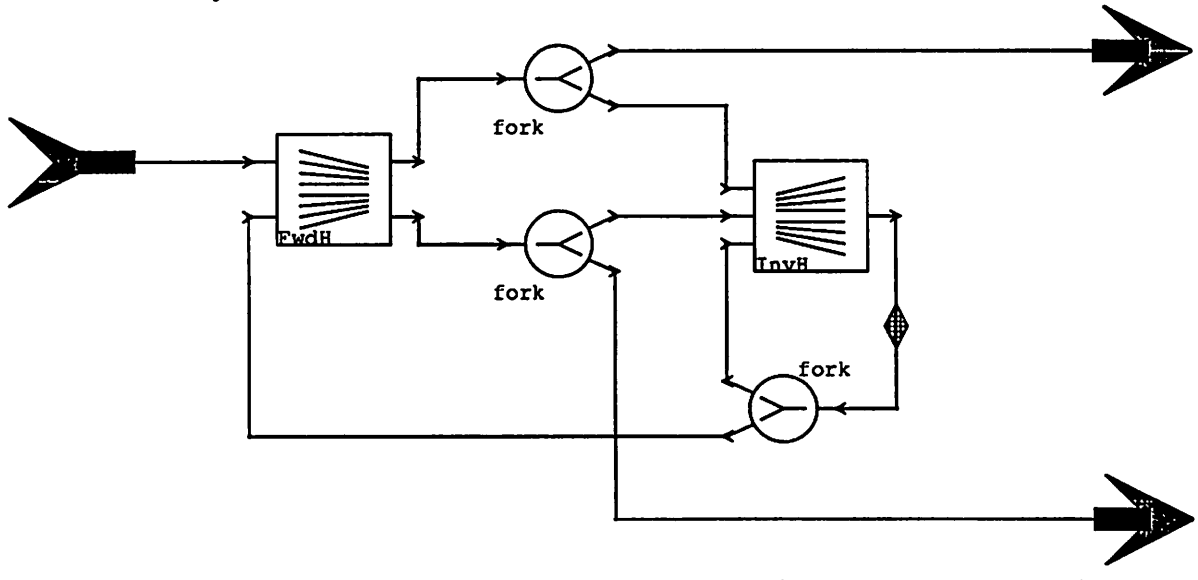
At every level of hierarchy, interconnected blocks in Ptolemy are executed in a particular *domain*. Each domain decides the order in which blocks are run, handles data exchanges between blocks and other levels of hierarchy, and performs auxiliary actions such as automatic code generation or interaction with stand-alone simulators (for example for DSP integrated circuits or circuit description languages). The simulations in this paper primarily rely on the synchronous dataflow (SDF) domain and the discrete event (DE) domain. The SDF domain handles subsystems in which each block produces and consumes a fixed number of data samples on each invocation. Most signal-processing type algorithms exhibit this type of behavior. The block execution order of SDF subsystems only need be calculated once; this speeds the simulation of signal processing systems.

The DE domain is more useful for modeling data networks, in which data exchanges between blocks are impossible to predict before execution. The DE scheduler assigns time-

stamps to all data samples within its subsystem and executes the block with the oldest input data sample. The scheduler establishes a method for scheduling blocks with simultaneous inputs as well.

The ability to run different parts of a simulation in different domains while exchanging information among the domains seamlessly is Ptolemy's strongest point. During system prototyping, a designer can model a complicated subsystem in an efficient domain while controlling the top-level simulation from the most powerful domain. As the design continues, parts of the system can be modeled first at the functional level and later at the register level (with a hardware description language such as VHDL or Thor). Programmable DSP applications first can be simulated at the functional level, later can be tested with DSP simulators, and finally can be compiled and loaded onto actual hardware.

The Ptolemy model of the video coder used in this section is built in the SDF domain.



High-Level Diagram of Ptolemy Video Coder (Fig. 20)

The "FwdH" block reads current and past image frames and outputs motion vectors, motion-compensation prediction errors, and α values. The "InvH" block inverts these actions. It generates an approximate version of the current frame, given motion vectors, α values, the previous frame, and the motion compensation difference. In the coder, delayed outputs from the "InvH" block are used as the past-frame inputs to both the "FwdH" and

"InvH" blocks.

Figure 21 shows the contents of the "FwdH" block. This block contains subblocks that perform motion compensation, the DCT, reordering of DCT coefficients, and quantization and Huffman coding.

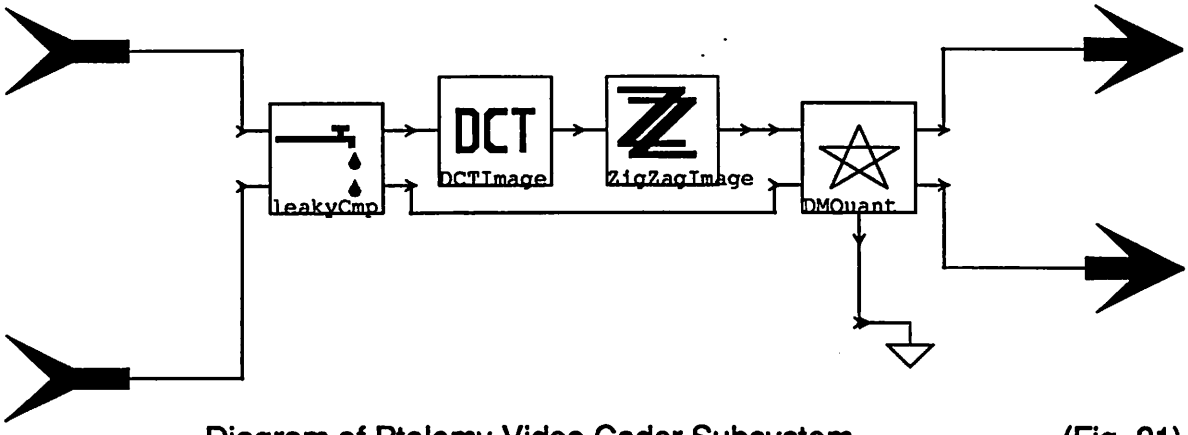


Diagram of Ptolemy Video Coder Subsystem (Fig. 21)

All of the parts of the coder, including the discrete cosine transform block, motion compensation block, and quantization blocks, read and write exactly one frame of video per invocation. Thus, Ptolemy can schedule the execution order and data transfer pattern of the blocks just once before executing any of the blocks. This eliminates overhead processing during coder simulations.

The coder is quite similar to that suggested by the H.261 and MPEG video coding standards. Interframe redundancy between successive frames is removed with motion-compensated prediction that uses a block size of 8 pixels and a fixed leak factor. Further, the coder uses a reduced-search motion estimation algorithm which is quite a bit more computationally efficient than full-search motion estimation. The reduced-search motion estimator searches an area that extends 15 pixels above, below, to the left, and right of each block in the previous frame. First, the 8 positions at offsets of ± 8 pixels from the original block are compared with the unshifted block. From whichever candidate block matches best, the 8 positions at offsets of ± 4 pixels are compared. From the winner of these comparisons, the blocks at offsets of ± 2 pixels are checked, and then the blocks at offsets of a single pixel.

We found that the loss in compressibility using reduced-search motion compensation as compared to full-search motion compensation is on the order of 1%.

Motion-compensated frame differences are transformed with an 8×8 DCT. Also, the first frame of each sequence and after every scene change is transformed directly with the DCT without the motion-compensated difference operation. The coder transmits one α value per frame: 0 if the frame comes right after a scene change and another fixed value otherwise. Intraframe coding of the frames after scene changes requires about 30% fewer bits than motion compensation coding.

The coder output is sent over three substreams. The frame-by-frame α values are transmitted over a "guaranteed-delivery" substream. If even a single α value is lost, all remaining video frames until the next scene change will be seriously in error. Motion vectors and DCT coefficients are quantized and sent over "high-priority" and "low-priority" substreams. For each block in the input sequence, the horizontal and vertical components of the block's motion vector are Huffman-coded and sent on the high-priority stream. Also, the brightness DCT coefficient is linearly quantized, Huffman-coded, and sent over the high-priority stream. There is nothing to be gained from sending motion vectors and DC coefficients over separate substreams since their combined rate characteristics are easily measured and since losses of either of these values have very similar effects.

All "AC" DCT coefficients from each block are scanned in a zig-zag pattern before quantization, which helps to increase the length of runs of zeros and thus improves the compressibility of the AC coefficients. Next, the coefficients are linearly quantized with a quantizer that has a dead-zone around 0. Runs of zeros are replaced with a symbol that marks the start of a zero-run and with the Huffman-coded length of the run. Nonzero AC coefficients are Huffman-coded; the lowest-frequency coefficients may be sent over the high-priority substream but the bulk are sent over the low-priority substream.

This video coder uses four separate Huffman code tables: one for the motion vector components; one for the DC DCT coefficients; one for the AC DCT coefficients, "start of

zero-run" marker, and a "start of block" marker; and one for the lengths of zero-runs. The symbol statistics for these tables were generated using a version of this video coder without the Huffman coding and with four test sequences. Before generation of the Huffman codes, the symbol statistics were smoothed so that very unlikely symbols would not have very long codewords.

There are a few possible improvements that could be made to the coder. The image noise level of non-motion-compensated blocks could be improved somewhat (or the bit-rate could be reduced) if higher-frequency AC DCT coefficients were quantized with a larger step size than lower-order coefficients. This frequency scaling degrades the image quality of predictively coded frames so it is not included. The compression ratio also could be improved if Huffman coding were done on pairs of an AC DCT coefficient and the length of the zero-run that follows each coefficient. (If another nonzero coefficient follows a particular coefficient, the zero-run length is 0.) MPEG, JPEG, and H.261 all use this compression scheme. These standards also allow the DC and AC coefficient quantizer parameters to be modified within an image. This can reduce the image noise level or reduce the bit-rate of sequences that are very different from those used to generate the initial Huffman tables, but we have found that it does not produce any notable benefit for numerous natural-scenery video clips of people, sports, and outdoor scenes.

Also as mentioned previously, full-search motion compensation would give a slightly lower bit-rate for a given image quality than reduced-search motion compensation. An additional method for improving the bit-rate/quality trade-off of this video coder would be to use sub-pixel motion estimation, in which virtual pixel values are calculated by interpolation for positions between the real pixels in the reference image, and motion compensation is performed using both the real and virtual pixels. Sub-pixel motion compensation might improve the coder performance significantly.

The image fidelity and bit-rate of this coder output depend on several parameters. The step sizes and dead-zone sizes for the AC and DC DCT coefficient quantizers determine

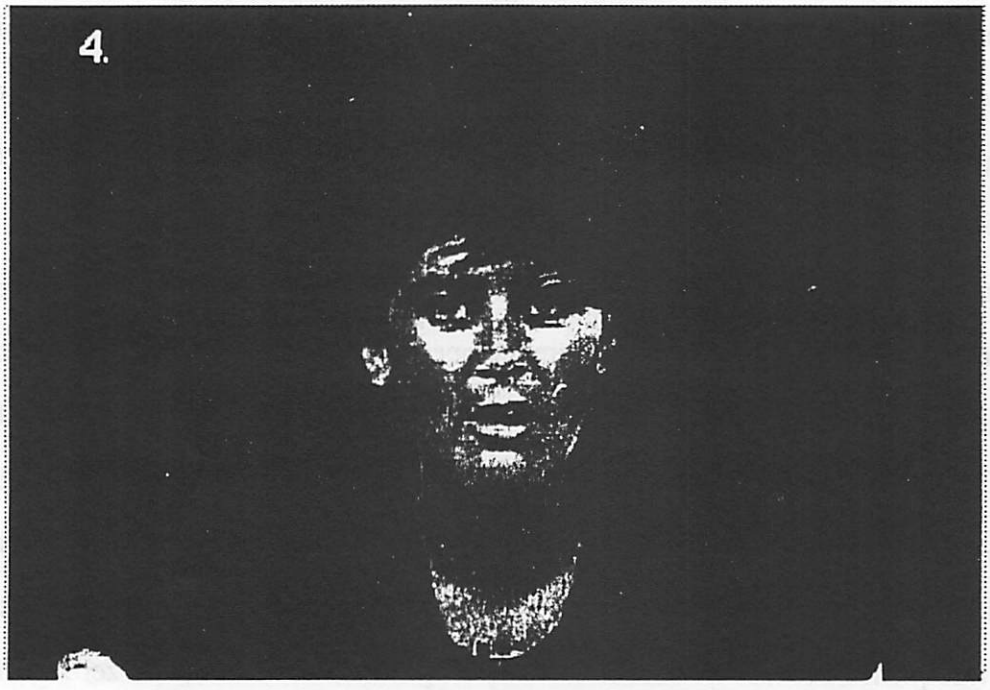
how noticeable noise and contouring artifacts are in the video output. Typically these parameters are chosen through subjective testing so that most people just cannot notice any artifacts. Then any reduction in these parameters increases the video bit-rate with no reduction in perceived image noise level, while any decrease quickly increases the amount of visible noise.

For channel setup negotiations we vary several other coder parameters. Changing the α value in the motion-compensation difference operation allows the coder to trade between higher bit-rates and increased protection from cell loss artifacts. The coder varies the number of DCT coefficients sent at high priority also; moving more coefficients to the high-priority substream alters the relative bit-rates on the high- and low-priority substreams and also allows more losses on the low-priority substream. This move also changes the total bit-rate of a sequence because a coder cannot run-length encode consecutive zero DCT coefficients that are sent on both substreams.

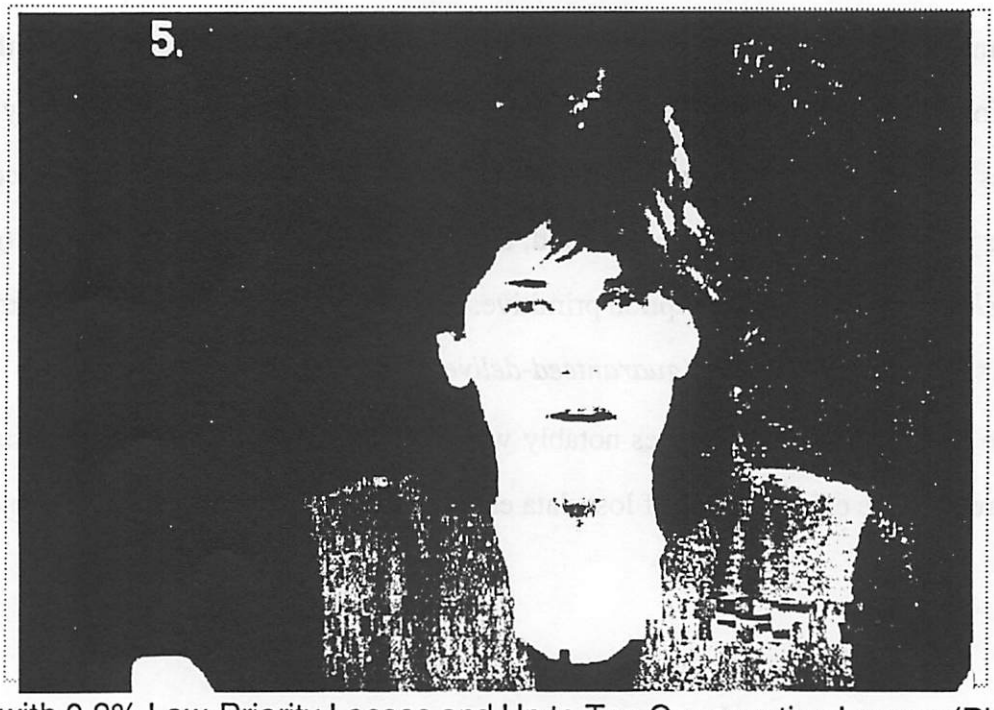
Many of the three substream parameters are fixed. All substreams must have the same delay for the receiver to work properly; we choose a value that gives sufficiently fast responses to user control actions. The rate of the guaranteed-delivery stream is fixed at one cell per frame time, or 30 cells per second; this rate can be specified with an RLB(*guaranteed-delivery*, 1, 30) rate description primitive. No losses are allowed on this stream, which can be specified with an LLB(*guaranteed-delivery*, 0, 1) loss primitive.

The coder performance varies notably with the burstiness of cell losses. Consecutive losses reduce the effectiveness of lost-data estimation in the receiver and also subjectively

make errors more noticeable.



Video with 0.2% Low-Priority Losses and No Consecutive Losses (Photo 4)



Video with 0.2% Low-Priority Losses and Up to Ten Consecutive Losses (Photo 5)

Of course the coder performance varies with the long-term average cell loss rate also. Any losses on the high-priority substream cause objectionable artifacts, so we choose a loss rate

for that stream that makes such defects suitably rare. We could use a cell loss bound of the form LLB(*high-priority*, 100, 10^7) to ensure that such losses occur on average about once per hour (10^7 cells delivered per loss / 10^4 cells per second = 10^3 seconds per loss). Losses on the low-priority substream occasionally are noticeable but rarely are truly objectionable; this substream can tolerate a much higher loss rate. We describe losses on the low-priority substream with two bounds: LLB(*low-priority*, M, $2M$) controls consecutive losses, and LLB(*low-priority*, 50, L) controls the loss rate at a larger timescale. Several simulations with different network models have shown that for L chosen such that $1/L \geq$ the long-term average loss rate, a leaky bucket of size 50 is not violated even with long loss bursts. During negotiations we vary L and M.

The data rates of the high- and low-priority substreams can be specified with leaky bucket bounds also: RLB(*high-priority*, 150, J) and RLB(*low-priority*, 150, K). Simulations with this coder indicate that with J and K approximately equal to the long-term average cell rates on the high- and low-priority substreams, monitors that allow bursts of up to 150 cells will not be violated even during scene changes. J and K vary during negotiations also.

As in section 3.5.4, we use numerous subjective tests to establish a set of channel flowspec parameters that yield the same perceived level of video quality—that is, a performance level-set. The parameters are listed in table 3 at the end of this chapter. The performance level-set contains a range of data rates and loss characteristics. We use the data in table 3 to conduct call setup negotiations as was done in section 3.5.4; that section's negotiation method can operate with a subjectively defined performance level-set. We use a channel cost function similar to those used in the negotiations in sections 3.5.3 and 3.5.4.

$$cost = \alpha J^{\gamma} + \beta K^{\gamma} + \kappa(\log(L) - \log(6.67)) + \lambda/(M + \mu)$$

This function charges for bandwidth through the J and K terms, buffer resources through the L term, and control effort through the M term. During negotiations we must divide all L values by 100, renormalizing to reduce the accuracy necessary for the calculations.

First we study a network in which bandwidth costs dominate buffer space costs. For a network with $\alpha = 2.0$, $\beta = 1.0$, $\gamma = 0.7$, $\kappa = 0.03$, $\lambda = 0.1$, and $\mu = 3.0$, the following table shows negotiation results for three sets of starting channel parameters.

initial parameters: *J, K, L, M*	initial cost	iterations	final parameters: *J, K, L, M*	final cost
5.80, 5.17, 400, 5.00	10.1	100	3.15, 5.60, 535, 4.47	7.96
3.22, 5.64, 500, 5.00	8.04	100	3.08, 5.46, 577, 4.64	7.82
10.1, 4.79, 167, 10.0	13.2	32	3.66, 6.31, 227, 9.9	8.71

Negotiations do not converge to the same final parameters because the iterations frequently carry intermediate flowspec parameter points away from the points that define the parameter level-set. However, final cost values are comparable. As expected because of the relatively low cost of buffer space, bandwidths and allowed loss rates both decrease during all three negotiations.

Results are more favorable with a different network that charges less for bandwidth and that does not impose a cost penalty for the transmission of high-priority data. Such a network could use a buffer management discipline that provides multiple levels of loss discard priority with little network processing effort (section 5.2.2). This network cost function uses $\alpha = 0.5$, $\beta = 0.5$, $\gamma = 0.7$, $\kappa = 3.0$, $\lambda = 10.0$, and $\mu = 3.0$.

initial parameters: *J, K, L, M*	initial cost	iterations	final parameters: *J, K, L, M*	final cost
5.80, 5.17, 400, 5.00	17.1	29	5.76, 5.45, 7.26, 5.04	5.16
3.22, 5.64, 500, 5.00	17.3	24	3.56, 6.08, 297, 5.35	15.9
10.1, 4.79, 167, 10.0	14.8	15	10.1, 5.25, 9.00, 10.0	6.10

The negotiation results again differ somewhat because negotiations carry intermediate flowspec parameter points away from the points that define the performance level-set for this application. Negotiations with this network achieve up to a 70% reduction in channel

cost, however! Below are shown frames from sequences that use the initial and final channel parameters in the last negotiation above.

Frame Transmitted with Initial Channel Parameters (Photo 6)

Frame Transmitted with Final Channel Parameters (Photo 7)

The starting point of this negotiation used a leaky motion compensation α value of 0.75

and 6 high-priority DCT coefficients per block. After negotiations the coder uses $\alpha = 0.5$ and 8 high-priority coefficients per block. These coder parameters can be derived from the coder parameters of points from the performance level-set near the final negotiated flowspec parameters. After negotiation, the coder sets its coder parameters appropriately, accepts a network channel with the negotiated flowspec parameters, and begins communications. The first information that must be sent to the video receiver is the coder parameters chosen.

The network's adaptation to the negotiated flowspec is discussed further in chapter 5. In particular, the network might establish a channel with a special buffer access discipline that limits the number of consecutive cell losses. Without this control, negotiations must assume that 15 or more cells in a row could be lost. The data in table 3 shows that, at least for leaky motion compensation α values close to 1 and 1 or 3 high-priority DCT coefficients per block, consecutive loss control allows 3 to 5 times higher loss rates. This allows 17% to 27% smaller buffer allocations for the video coder, using the model that buffer requirements are proportional to $\log(L)$, where L is the token parameter in the leaky bucket loss bound.

If this video coder supported audio transport also, parameters of the audio substreams could be negotiated separately from those of the video substreams. The application would use a single channel that includes both video and audio substreams, thus simplifying network management, routing, and synchronization of the audio and video transport. During negotiations, parameters of the video substreams could be traded against each other and parameters of the audio substreams could as well. Unless the application knows how to trade between video and audio performance however, it could keep both performance levels fixed by not allowing trade-offs between video and audio substream parameters. This essentially decouples a constant-performance surface that includes both video and audio parameters into independent surfaces for each.

4.6 Other Video Coding Methods and the Medley Interface

Below, we present some video applications that become feasible or more efficient with flexible networks such as those employing the Medley Interface. Video coders, and more so multimedia coders, benefit from having multiple substreams with different quality of service (QOS) specifications available because these applications often produce several different data types with different effects on their perceived qualities.

4.6.1 Improved Video Compression Methods

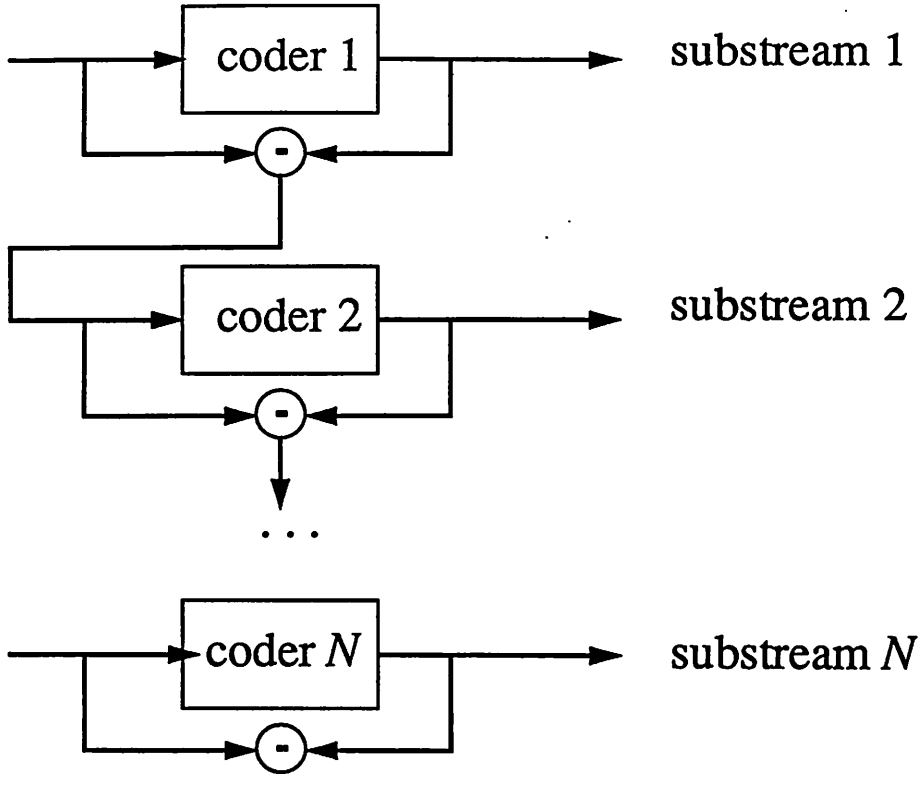
In addition to broadcast video, many new communications applications using video are being studied and tested. For example, commercial two-way videoconferencing systems currently are available, and researchers are beginning to plan for portable hand-held video communications devices [115]. Also, computer manufacturers are improving their display hardware and control software for true multimedia applications. Large-scale video databases such as in the Sequoia project [117] only will be useful if the stored video can be accessed and viewed remotely by several users at once.

These new video applications as well as existing video coding methods can be extended to offer improved performance with flexible networks. For each application, the designer must identify the transport needs for different data types and must decide how different transport resources can be used to support the application.

4.6.1.1 Multiresolution and Progressive Video Coders

The first stage of a multiresolution coder represents and outputs a coarse representation of its input image. The next stage codes more finely the difference between the original input and the first stage's coarse representation. Each stage of the multiresolution coder represents (somewhat imperfectly) the coding error of the previous stage. Thus, as the receiver adds together more outputs from a multiresolution coder, its received image becomes more

accurate.



Multiresolution Coder (Fig. 22)

The Medley Interface model works well with multi-resolution coders. Each of multiple receivers connected to a single coder can receive only the substreams necessary for its desired performance level. Also, since the Medley Interface supports more than two levels of loss priority, each of many multi-resolution substreams can be protected more than higher layers; this makes sense since each layer is useless unless all lower layers are received correctly.

Progressive coders also divide video into one coarse and several finer approximations. The coarsest data are transmitted quickly and later approximations are transmitted more slowly and more reliably. Progressive transmission systems differ from multi-resolution systems in that their high-delay approximations must be accurate even if coarser approximations are corrupted by data losses. Progressive systems efficiently allow users to browse through many images and to see only a few at fine resolution.

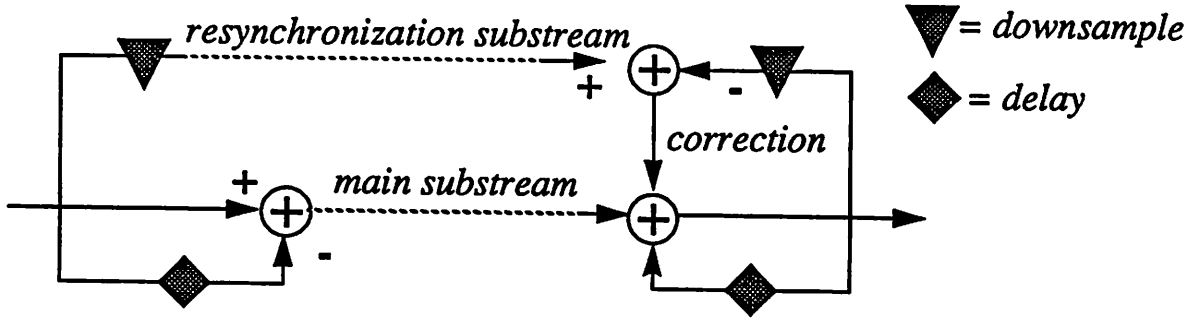
When a progressive coder sends data sequentially over a single homogeneous channel, the coder must store at least an entire frame of data. At each progressive coding step, the coder must have available the difference between the original frame and the most recent approximation. The coder then codes the remaining error approximately, transmits the approximation, and stores the error of the newest approximation.

With a Medley Interface network, a source could code subregions of each image (such as blocks) with a multiresolution technique and then send the different multiresolution descriptions over different substreams with different delays. As the destination receives and decodes new data, it simply adds the decoded values to the correct part of the picture frame.

4.6.1.2 Loss Recovery

Previous research that studies how to limit error propagation caused by cell loss depends a great deal on the percentage of video data that can be sent at high-priority [7, 16, 17]. In real networks, if a significant fraction of coded video data is sent at high priority, then either transmission costs become high or some cells initially sent at high priority get bumped to low priority. If even a few high priority cells suffer as much loss and delay as low priority cells, then video quality at the receiver suffers drastically.

Modified video coding methods such as discussed in section 4.4 allow better performance. Another coding method possible with Medley Interface networks is to send compressed video over several substreams as usual, and then to send resynchronization data over a cheap, low-bandwidth guaranteed-delivery substream (that might use an automatic repeat request protocol). A receiver never displays the data on the resynchronization sub-



stream because it often arrives too late to be useful, but does use the data to correct accumulated errors in the displayed information. For any replenishment data to be useful to a video receiver, the receiver must store a copy of the video data that it actually displayed at the time the replenishment data were generated. The difference between the displayed and correct data presumably is due to transmission errors—this difference is subtracted from the currently displayed frame to eliminate the effects of the errors.

4.6.1.3 Transform Coding on Non-Rectangular Shapes

Transform coding of nonrectangular picture regions is discussed in [25]. Such systems must transmit shape information in addition to transform coefficients. Since the transform coefficient values are useless without correct shape information, this data should be sent more reliably than the coefficients. Sill, there is a benefit to sending low-frequency coefficients over more reliable substreams than high-frequency coefficients. A flexible network can support the description of these loss needs as well as different rate characteristics of the different data types.

4.6.1.4 Direction-Adaptive Subband Coding

Directional subband coders, which perform transform coding along the direction of motion in a video scene, may offer more compression potential than the DCT [31]. It is critically important that these coders receive the direction vectors for each block accurately. A channel with multiple-discard priorities with a lossless substream could support this application well.

4.6.2 Multimedia

Multimedia applications especially benefit from flexible networks since multimedia traffic streams contain substreams with widely varying requirements: video, audio, image, graphics, procedures, text, control information, etc. Audio and video data have fairly strict loss and delay requirements but different bandwidth needs. Images, graphics, and text can tolerate higher delays but must be transmitted 100% reliably. Procedural and control information may or may not have real-time delivery constraints, but they also must be delivered

without error.

A single homogeneous channel for a multimedia application would need to meet the most stringent transport requirements of all of these data types. Such a high-bandwidth, low-delay, lossless channel would be expensive. By supporting the transport needs of each data type separately, a network could support the application at the same perceived performance level while consuming many fewer resources.

4.7 Conclusion

This chapter discusses how video coders based on motion compensation and the DCT can provide high-quality video with a range of network channels. Conditional replenishment, leaky motion compensation, and conditional leaky motion compensation give such video coders protection from cell loss artifacts for only moderate bandwidth penalties. They limit error propagation such that errors are corrected faster and less obtrusively than in periodic replenishment coders. Leaky motion compensation is used by the video coder prototype that negotiates with several Medley Interface networks to obtain low-cost channels for high performance—simulated negotiations reduce channel costs by up to 70%.

The end of this chapter reviews other video coding techniques that would benefit from networks with multiple tunable substreams. Video as well as other signal types often are encoded in such a way that transmitted data values vary widely in their rate, delay, and loss sensitivity characteristics. By sending different data types over substreams tailored to their needs, an application can use network resources more efficiently than if they all were sent over a homogeneous channel.

4.8 Appendix

The table below lists channel flowspec parameters for the video coder described in section 4.5 that subjective tests indicate yield video sequences of roughly constant performance. We first picked a baseline set of flowspec parameters that produced video with barely perceptible coding artifacts. Then, for a range of flowspec parameters, we asked

graduate students to compare the quality of the baseline video sequence with test sequences. Flowspec parameter sets that produced test sequences of equivalent quality to the baseline sequence are listed below.

α	DCT coefficients on high-priority substream	*J*, high-priority kcells/sec	*K*, low-priority kcells/sec	*L*, cell deliveries per loss	*M*, consecutive cell losses
0.9375	1	2.964	5.486	667	1
0.9375	1	2.964	5.486	1000	5
0.9375	1	2.964	5.486	1667	10
0.9375	1	2.964	5.486	3333	15
0.9375	3	5.772	5.226	500	1
0.9375	3	5.772	5.226	667	5
0.9375	3	5.772	5.226	1000	10
0.9375	3	5.772	5.226	2000	15
0.9375	6	10.14	4.862	400	1
0.9375	6	10.14	4.862	400	5
0.9375	6	10.14	4.862	500	10
0.9375	6	10.14	4.862	667	15
0.9375	10	16.12	4.394	50	1
0.9375	10	16.12	4.394	50	5
0.9375	10	16.12	4.394	66.7	10
0.9375	10	16.12	4.394	100	15
0.875	1	3.068	5.486	500	1
0.875	1	3.068	5.486	667	5
0.875	1	3.068	5.486	1333	10
0.875	1	3.068	5.486	2500	15
0.875	3	5.798	5.174	333	1
0.875	3	5.798	5.174	400	5

α	DCT coefficients on high-priority substream	*J*, high-priority kcells/sec	*K*, low-priority kcells/sec	*L*, cell deliveries per loss	*M*, consecutive cell losses
0.875	3	5.798	5.174	667	10
0.875	3	5.798	5.174	1000	15
0.875	6	10.114	4.784	222	1
0.875	6	10.114	4.784	250	5
0.875	6	10.114	4.784	286	10
0.875	6	10.114	4.784	333	15
0.875	10	16.094	4.316	33.3	1
0.875	10	16.094	4.316	40	5
0.875	10	16.094	4.316	50	10
0.875	10	16.094	4.316	66.7	15
0.75	1	3.224	5.642	400	1
0.75	1	3.224	5.642	500	5
0.75	1	3.224	5.642	1000	10
0.75	1	3.224	5.642	2000	15
0.75	3	5.85	5.252	200	1
0.75	3	5.85	5.252	250	5
0.75	3	5.85	5.252	333	10
0.75	3	5.85	5.252	500	15
0.75	6	10.088	4.784	125	1
0.75	6	10.088	4.784	143	5
0.75	6	10.088	4.784	167	10
0.75	6	10.088	4.784	200	15
0.75	10	15.964	4.264	12.5	1
0.75	10	15.964	4.264	12.5	5
0.75	10	15.964	4.264	12.5	10

α	DCT coefficients on high-priority substream	*J*, high-priority kcells/sec	*K*, low-priority kcells/sec	*L*, cell deliveries per loss	*M*, consecutive cell losses
0.75	10	15.964	4.264	12.5	15
0.5	1	3.458	5.98	250	1
0.5	1	3.458	5.98	333	5
0.5	1	3.458	5.98	500	10
0.5	1	3.458	5.98	667	15
0.5	3	5.98	5.486	111	1
0.5	3	5.98	5.486	125	5
0.5	3	5.98	5.486	143	10
0.5	3	5.98	5.486	200	15
0.5	6	10.088	4.94	83.3	1
0.5	6	10.088	4.94	100	5
0.5	6	10.088	4.94	100	10
0.5	6	10.088	4.94	125	15
0.5	10	15.86	4.342	6.67	1
0.5	10	15.86	4.342	6.67	5
0.5	10	15.86	4.342	6.67	10
0.5	10	15.86	4.342	6.67	15
0.0	1	3.458	11.96	333	1
0.0	1	3.458	11.96	400	5
0.0	1	3.458	11.96	667	10
0.0	1	3.458	11.96	1000	15
0.0	3	5.72	10.92	200	1
0.0	3	5.72	10.92	250	5
0.0	3	5.72	10.92	333	10
0.0	3	5.72	10.92	500	15

α	DCT coefficients on high-priority substream	*J*, high-priority kcells/sec	*K*, low-priority kcells/sec	*L*, cell deliveries per loss	*M*, consecutive cell losses
0.0	6	9.282	9.75	125	1
0.0	6	9.282	9.75	125	5
0.0	6	9.282	9.75	167	10
0.0	6	9.282	9.75	167	15
0.0	10	14.248	8.502	10	1
0.0	10	14.248	8.502	10	5
0.0	10	14.248	8.502	10	10
0.0	10	14.248	8.502	10	15

Performance Level-Set for the Motion Compensation + DCT Video Coder (Table 3)

Chapter 5

BUFFER MANAGEMENT DISCIPLINES FOR FLEXIBLE NETWORKS

Chapter 3 presented a network interface model that allows communications applications to express their rate and network quality of service (QOS) needs fairly exactly. Within this model, several different applications have negotiated with high-speed networks to obtain low-cost channels that support the applications' needs. The previous chapter discussed how a modern video coder could adapt to function well with the variety of channels that might result from negotiation. Without such adaptability, applications only could operate with a very limited set of flowspec parameters. This chapter discusses the opposite problem—how networks can adapt to the specialized needs of their client applications.

Good buffer management strategies are vital to the provision of high network QOS with high resource utilization. Networks must choose their buffer management disciplines to ensure that application data are transported quickly enough and are lost rarely enough to satisfy their clients. Buffer management conceptually consists of two interrelated disciplines: buffer service and buffer access. A buffer service discipline determines the order in which stored cells are removed from one or more buffers before transmission over an output link. Since a buffer's service discipline controls how long an arriving cell must await retransmission, it determines the delay and bandwidth characteristics of channels that pass through the buffer. A buffer access discipline determines whether to store or discard cells that arrive at one or more buffers. Since the buffer access discipline controls a switching node's cell discard behavior, the buffer access discipline determines the loss characteristics of channels that pass through the buffer. When applications transmit different data types over a single channel, frequently the applications find it useful to specify that some data is more

sensitive to loss than the rest. A network that supports multiple levels of *loss priority* must use a buffer access discipline that selectively discards low-priority data before high-priority data.

Buffer service disciplines for various systems have been studied in [35, 45, 53, 64, 75, 85]. Since cell-relay networks traditionally have been used to support data-transfer applications and protocols that detect and retransmit lost data, perhaps it is reasonable that less attention has been given to the control of cell loss rates than to bandwidth and delay characteristics. However, applications that require steady streams of data, such as audio and video, cannot tolerate the delay and processing overhead inherent in automatic repeat request (ARQ) protocols. Therefore, designers of networks that support these applications must give careful thought to the design of buffer access disciplines.

We have identified two types of applications that benefit from the control of channel loss characteristics other than the channel's average loss rate. Applications such as file-transfer and low-bit-rate voice must resynchronize the transmitter and receiver after any cell loss. The subjective performance or throughput of these applications depends more strongly on the expected time between losses than on the probability of cell loss. Losses can occur in long consecutive bursts as long as the bursts are infrequent. Oppositely, applications such as high-quality video that can estimate lost data perform best if consecutive cells are not lost; consecutive losses reduce the accuracy of the estimates. These applications' subjective performance depends on both their channels' average loss rate and on the distribution of the lengths of cell loss bursts.

This chapter studies the design of buffer access disciplines that provide a range of cell loss burstiness characteristics. Disciplines well-suited for file-transfer applications concentrate cell losses together; those well-suited for video applications spread out losses. The benefits of these disciplines lead us to propose that networks support a variety of buffer management disciplines in their switching nodes, so that the loss and delay characteristics of specific channels can be tuned for the needs of the channels' client applications. Differ-

ent buffers within a switch could implement application-specific disciplines, or special disciplines could be developed that support simultaneously the needs of several applications. As seen in the simulations and analyses of new buffer access disciplines in the following sections, with buffer access disciplines tailored for applications' needs, networks can allocate less buffer space and bandwidth to achieve the same QOS.

A network of course must know application needs to tailor buffer management disciplines to them. Applications can specify their loss rate, spacing, and priority requirements using the Medley Interface flowspec description format described in chapter 3. Just as the Medley Interface facilitated flowspec parameter negotiations and video coder adaptations, it enables networks to use specialized buffer management disciplines to implement channels with smaller resource requirements than would be required with generic disciplines.

5.1 Buffer Access Disciplines

Works that analyze cell-relay network buffer disciplines include the control of high- and low-priority losses in shared buffers [64], queueing control for minimum network delay [75], a queue control method that supports both low-loss and low-delay traffic [35], and a review of a number of buffer management disciplines [85]. We know of no works that study loss burstiness or that advocate that networks adapt their buffer management disciplines to their channels' specific needs.

Any buffer access discipline is either *work-conserving* or *non-work-conserving*. Work-conserving buffer access disciplines never discard cells when there is buffer space available to store them [64]. Thus, all have the same queue length distribution and overall cell loss probabilities when fed the same arrival stream. Work-conserving disciplines can differ in their choice of cells to discard when inputs arrive at a full buffer, producing channels with different degrees of loss burstiness or loss priority protection.

Non-work-conserving buffer access disciplines can discard cells even when buffer space is available. Such discards free space that might be more useful in the future. For example, a queue that serves alternating bursts of high- and low-priority cells could discard

low-priority data from a partially filled queue in anticipation of a high-priority burst. As with work-conserving disciplines, the rules that non-work-conserving disciplines use to discard cells can be tuned to achieve different QOS objectives. For some QOS objectives non-work-conserving disciplines are simpler to implement than work-conserving disciplines (section 5.2).

While loss characteristics are arguably the most important component of network QOS for real-time applications such as video and audio, the average loss rate is not the most important loss statistic for all communications applications. As mentioned previously, compressed video applications are very sensitive to the loss of several cells in a row. File transfer applications need long periods uninterrupted by cell losses, although they can tolerate long bursts of missing cells when losses do occur. The cell discard rules practiced by different work-conserving or non-work-conserving buffer access disciplines can be designed to meet these different loss burstiness needs. Section 5.2 reviews disciplines that regulate the treatment of cells with different discard priorities. Section 5.3 reviews an existing buffer access discipline that maximizes the time between loss bursts, and section 5.4 extends that technique. Section 5.5 presents a new discipline that minimizes consecutive cell losses.

5.2 Loss Priority Control

The simplest buffer access discipline, *first-come-first-served* (FCFS), directs that whenever a cell arrives at a queue with empty slots, the cell is placed at the end of an ordered list. When an input arrives at a full queue, it is discarded. The queue outputs elements from the front of the list. Since the FCFS discipline only discards cells when the buffer is full, this discipline is work-conserving.

The FCFS discipline is easily modeled with a Markov chain. Each state of the chain corresponds to a particular level of queue occupancy, and given the probability distribution of arrivals, the Markov transition probabilities can be derived.

Although FCFS is easy to implement, it makes no provisions for handling inputs of dif-

ferent priorities. We now review two methods that protect higher priority data against losses more than lower priority data.

5.2.1 Buffer Pushout

A simple modification of the FCFS discipline that allows for cells of different priorities is called *FCFS with pushout* [64]. This buffer access discipline works much like FCFS, but when a cell arrives at a full queue, the FCFS with pushout discipline discards a previously queued cell of lower priority. Most sensibly, the queued cell with lowest priority is dequeued and discarded. FCFS with pushout is a work-conserving buffer access strategy. Thus, it has the same cell loss rate as ordinary FCFS. However, FCFS with pushout gives communications applications more control over which cells are lost.

FCFS with pushout queues are more complicated to model than FCFS queues. Since this discipline mixes inputs of different priorities in the same queue, the priority of each queue entry must be known in order to calculate loss probabilities for each priority level. To model this information with a Markov chain would require s^N states, where s is the number of priorities and N is the queue length. Even for two priorities and moderate queue lengths, this approach is computationally infeasible.

A discipline in which arrivals of different priorities are stored in different queues and higher-priority queues are always served before lower-priority queues is simpler to analyze. However, this is not as useful for video traffic because the delays suffered by low-priority data are much longer than those suffered by higher-priority data, given comparable arrival rates.

5.2.2 Partial Buffer Sharing

Next, we review the *partial buffer sharing* discipline [64]. This discipline prevents lower-priority cells from entering a queue as the queue nears capacity. A partial buffer sharing queue of length L can be described with a *screening function* $r_L(l)$ that specifies the minimum priority level that a cell must have to gain admission to a queue of length L when that

queue contains l cells.As l increases, cells need higher priority levels to be admitted to the queue. $r_L(0)$ should equal the lowest priority in the system, and $r_L(L)$ should be higher than the highest priority in the system. For example, in a system in which priorities range from 0 to p_{MAX}, $r_L(l)$ could be the function $max(0, p_{MAX} - L + l)$ or $\frac{l \cdot p_{MAX}}{L}$.

Partial buffer sharing differs from FCFS with pushout in that partial buffer sharing does not queue some low-priority arriving cells even when the queue is able to store them; the queue space is reserved for future higher-priority arrivals. Thus priority screening is not work-conserving. However, partial buffer sharing is simpler to implement than FCFS with pushout because the priorities of the queue contents need not be examined with partial buffer sharing. With both schemes, when a queue is nearly full then only the highest-priority traffic is delivered, as desired. Also, both schemes decide whether or not to admit a cell based solely on the current state of the queue.

In a FCFS with pushout buffer, no high-priority cells are discarded until all low-priority cells have been pushed out. Thus, the low-priority traffic cannot affect the loss rate of the high-priority traffic. Of course, the amount of high-priority traffic affects the loss rate of the low-priority traffic significantly. For some applications, this behavior is ideal. For example, in a still image transmission application based on the JPEG standard [103] the brightness discrete cosine transform (DCT) coefficients would be sent at high priority and the detail coefficients at low priority. The loss of any brightness coefficients adversely affects the received image quality so much more than the loss of detail coefficients that a cell containing detail coefficients *never* should be delivered instead of a cell containing brightness coefficients.

However, for other applications this behavior is too drastic. High-priority traffic should experience fewer losses than low-priority traffic, but both priority classes should satisfy certain loss probability bounds. A network switch could discard a high-priority cell if the discard would not violate the loss bound for the high priority class. An application that combines video and audio transmission might benefit from this sort of network behavior.

Audio cells are sent at high priority and video cells are sent at low priority because audio quality is more important than video quality for this particular application. However, the loss of a small number of cells of either audio or video can be mitigated with proper estimation techniques. As long as the audio cell loss rate is low enough, audio cells can be discarded to maintain high video quality.

Partial buffer sharing provides tunable loss probabilities among all input priority classes through proper design of the function $r_L(l)$. With only two priority classes the selection of a screening function is fairly straightforward; the only adjustable parameter is the length at which the queue stops accepting low-priority cells. However, the design of a screening function for a system with many input priority classes is difficult. The function $r_L(l)$ should increase monotonically, since the more queue slots a given priority class is allowed to enter, the lower is its loss probability (as long as arrivals of each priority class are independent of the queue state). As the cutoff is lowered for a particular priority class, that class' loss probability increases but the loss probabilities of all other classes decrease; the amounts of the changes depend on the source statistics.

A queue could use an adaptive screening function that adapts for local changes in the input statistics. Such a queue updates its estimates of the arrival probabilities of each different priority class. Then, the queue can calculate a screening function to satisfy specified loss bounds or simply can adjust its current screening function to try to minimize the loss probability of high-priority inputs.

With multiple priority classes there can be no "optimal" buffer access strategy for minimum loss probabilities. In general, as a buffer access discipline is altered to provide a lower probability of loss for high-priority traffic, the loss probability for lower-priority traffic increases. The designer of an application must choose loss probabilities for each priority class and then must try to find a buffer access strategy that supports that quality of service.

Partial buffer sharing is somewhat easier to study analytically than FCFS with pushout since cells are only rejected at the time of their arrival, and whether or not they are rejected

depends only upon the instantaneous queue state. One case that can be analyzed is that in which all input priorities have independent exponential distributions and service is independent and exponential as well. A Markov process tracks the queue length, and each transition corresponds to an arrival or a service. Transitions only occur to adjacent states, and the probability of transition is determined by the probability of service vs. the probability of arrival of any input that is not rejected at the current queue length. To find each priority's loss probability, we sum the probabilities of being in any state in which that priority is rejected.

$$Pr(\text{lose priority } p) = \sum_{i:\, p \text{ is lost}} Pr(\text{queue state} = i)$$

In this way, the loss probabilities of different priority traffic can be compared for different screening functions.

A partial buffer sharing queue with deterministic service can be analyzed in discrete time. A Markov chain tracks the length of a queue, and the chain transitions correspond to service times. A random number of arrivals from each priority class arrives every timestep. At each timestep, highest-priority cells are admitted to the buffer first until all highest-priority arrivals have been accepted or until the buffer fills. Then, if the second-highest priority is still accepted by the screening function, those arrivals are accepted until the buffer fills or until the screening function increases past the second highest priority. Lower priorities are admitted similarly. Transitions of the Markov chain can jump to the preceding state or to any higher state; a transition jumps to the preceding state only if no arrivals occur during the service interval. Given that the buffer is in a particular state, the loss probability of a cell with arbitrary input priority is somewhat tedious to calculate since it must be calculated conditionally on the number of higher-priority arrivals in the same timestep.

5.3 Queue Purging and Queue Flushing

One disadvantage of the above buffer access disciplines is that once a buffer is full it is possible that the buffer will remain full or nearly full for a long time, during which chan-

nels' loss probabilities will be high. Several buffer access disciplines forcibly reduce a buffer's occupancy in hopes of preventing long loss durations. The rationale for these disciplines is that a buffer overflow builds gradually, because many of the buffer's inputs are sending at or just over their allowed rates. If a buffer discards many inputs whenever it fills, the buffer will take a long time to fill again. Thus the time between errors may be kept lower with occasional bulk discards than without them.

A drastic buffer access discipline that tries to maximize the duration between cell loss bursts can be called *queue purging*. With queue purging, whenever a queue fills every queued cell is discarded. Although each purging results in a number of cell losses equal to the queue length, the fact that the queue is empty after the purge makes the expected time between purges long.

A better alternative, called *queue flushing* [81], specifies that when a queue fills, newly arriving inputs are discarded until the queue has emptied. This is less drastic than queue purging because the number of inputs that arrive during the time necessary to serve all queued cells is much less than the queue capacity. Otherwise, the queue service rate would be insufficient to handle even the expected queue traffic and the queue would overflow almost continuously.

5.4 Prioritized Queue Purging and Prioritized Queue Flushing

We next present some new buffer access disciplines, modifications of queue purging and queue flushing, that provide more control of channel loss characteristics. A new discipline called *prioritized purging* protects important traffic while maintaining an expected time between buffer overflows nearly as high as ordinary queue purging. With prioritized purging, when a buffer fills, only buffered cells from low-priority inputs are discarded. As long as most queued cells are low-priority then the buffer purge should prevent data discard for a substantial time. However, cells from high-priority sources only are lost if the queue fills with high-priority data. We can simulate this buffer access method, but to analyze it analytically is difficult since we would need to keep track of the position of each high-pri-

ority cell within a buffer.

Of course, queue flushing can be modified to handle multiple priority traffic in the same way as queue purging. With *prioritized queue flushing*, only low-priority arrivals at a queue that is flushing are rejected. With this scheme, a queue may never empty entirely or may take a long time to do so because of high-priority arrivals. Thus the queue might reject low-priority inputs for a long time. Some maximum duration of low-priority flushing could be enforced; the duration could equal the amount of time necessary to serve one queue-length's worth of cells, for example.

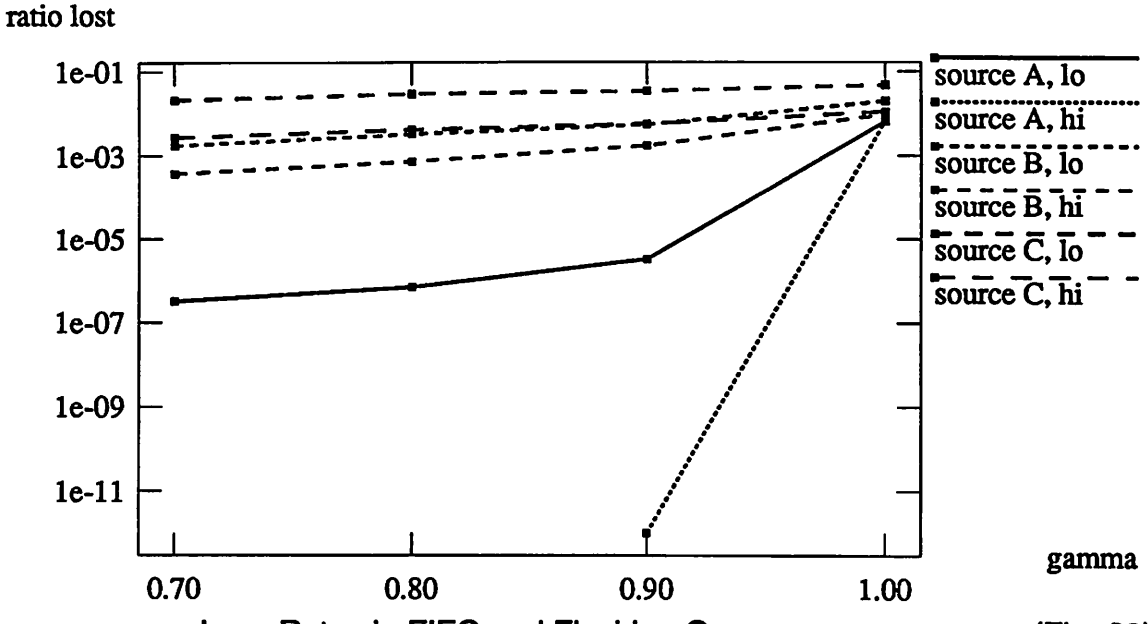
This discipline is superior to the prioritized queue purging discipline for the same reason that queue flushing is superior to queue purging. With prioritized queue flushing, the expected number of low-priority losses is much lower than with prioritized queue purging, but both flushing and purging have the ability to increase the time between cell loss bursts.

No buffer access discipline can maximize the time between cell loss bursts—disciplines can reject more and more arriving cells in a row to produce longer and longer times between bursts (as well as longer bursts). Practical disciplines must choose useful trade-offs between the average spacing between cell loss bursts and the overall cell loss rates. In fact, if prioritized queue flushing gives a particular channel too high a low-priority loss rate and more time than is needed between loss bursts, then *partial flushing*, in which arrivals are discarded until the queue empties only partially, gives fewer losses and less time between loss bursts.

To analyze prioritized queue flushing via a Markov chain is fairly straightforward if we do not impose a maximum duration of flushing. We assume that a queue that is flushing low-priority inputs continues to do so until the queue empties entirely. A Markov analysis of prioritized queue flushing is presented in the appendix.

Using that analysis we can find loss probabilities and expected times in non-flush mode for various buffer sizes and inputs. In figure 23, the input is a combination of 30 Bernoulli sources, each with probability of occurrence 0.03. The FIFO loss rates and loss burst rates

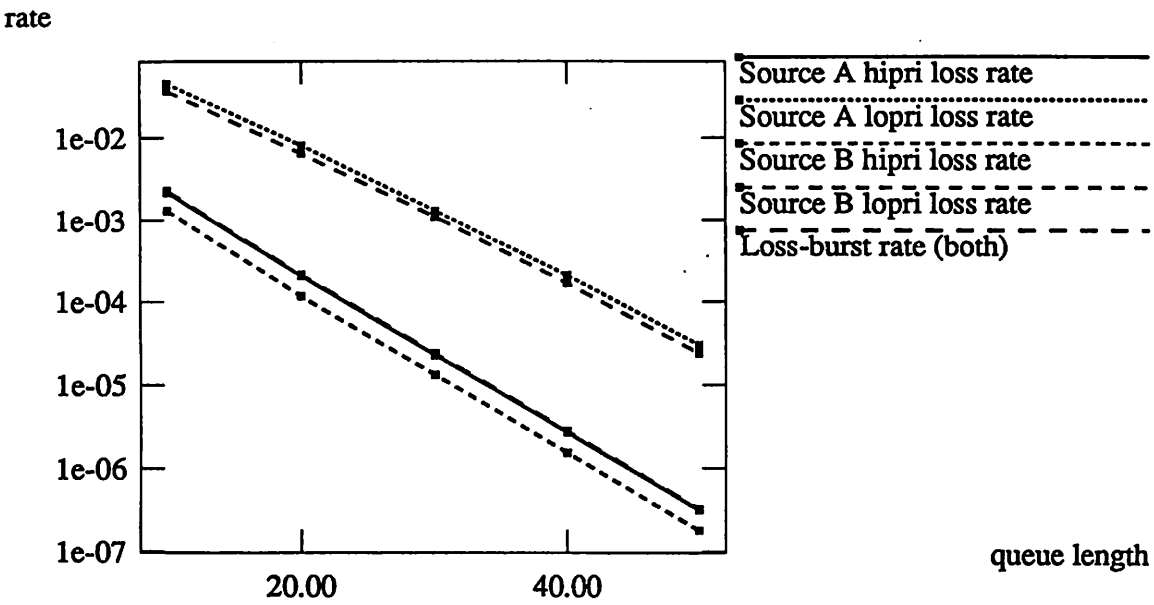
(i.e. rates at which one or more cells are lost in a row) are nearly equal. The flushing queue loses around an order of magnitude more cells than the FIFO queue, but many losses occur consecutively. Thus the flushing queue loss burst rate is about an order of magnitude smaller than the FIFO loss burst rate. These rate differences grow slightly more pronounced as queue lengths increase. As the flushing queue length grows, the queue suffers longer loss bursts, but it suffers them less often. Thus the loss characteristics of the flushing queue become less like that of the FIFO queue as the queue lengths increase.



Loss Rates in FIFO and Flushing Queues (Fig. 23)

Figure 24 contains data for two sources: source A consists of 15 high- and 15 low-priority Bernoulli(p = 0.03) inputs. Source B contains 10 high- and 20 low-priority Bernoulli(p = 0.03) inputs. For both sources, the low-priority loss rates are nearly equal. The high-priority loss rates for source B are somewhat lower than for source A, sensibly enough, since source B outputs less high-priority data. For both sources the loss burst rates are nearly equal since in either case once the flushing state is entered, it is left very quickly. Thus, the rate at which either source enters the flushing state is approximately the time required for 30 Bernoulli(p = 0.03) sources to fill an empty queue.

One interesting note is that the rate of entrance into the flushing state in the priority



Loss Rates in Partial Flushing Queues (Fig. 24)

flushing queue is slightly lower than this rate in the ordinary flushing queue. This is because the priority flushing queue spends slightly longer in the flushing state than the flushing queue because it accepts high-priority arrivals. This longer duration also contributes to the higher percentage of low-priority cell loss in the priority flushing queue than in the flushing queue. Another contribution is that when a combination of high- and low-priority arrivals cause the priority flush queue to overflow, all low-priority arrivals are discarded before any high-priority arrivals are lost. However, even for equal numbers of high- and low-priority arrivals, the low-priority loss probability in the priority flushing queue is only 1.8 times larger than the loss probability in the ordinary flushing queue.

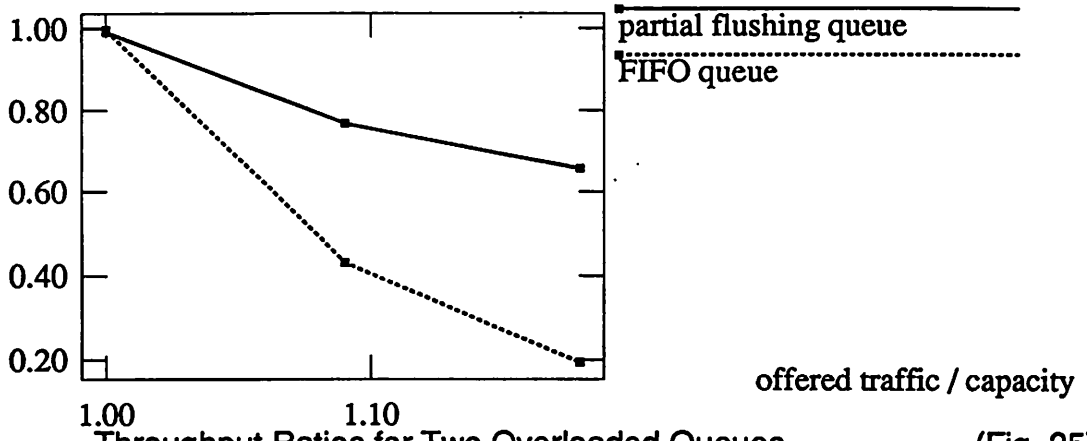
Even with equal numbers of high- and low-priority arrivals, the priority flushing queue gives high-priority arrivals a one or two order of magnitude smaller loss probability than low-priority arrivals. This spread becomes larger as the ratio of low- to high-priority traffic increases. This simple modification of queue flushing can extend significant protection to high-priority traffic.

In the above analyses, the flushing queue suffers a loss percentage 2 to 10 times higher than that of the FIFO queue. However, these losses are confined to the periods when the

flushing queue is actually flushing. As mentioned in the introduction, some communications applications are more adversely affected by a high frequency of loss bursts rather than by the length of the bursts. For example, we have simulated a packet-based file-transfer application that uses a sliding window protocol to implement reliable sequenced delivery of user packets (section 3.5.3). The protocol retransmits an entire packet whenever any cell in the packet is lost. A flushing queue can cause fewer packet errors than a FIFO queue, giving better performance to the file-transfer application.

The file-transfer application is simulated operating in parallel with several bursty cross-traffic sources. When the total amount of file-transfer traffic and cross traffic is less than the network capacity then no cell losses occur. However, as the amount of cross traffic increases and losses begin, a partial flushing queue supports a higher packet throughput rate than does a FIFO queue. The results in section 3.5.3 show that when a partial flushing queue and a FIFO queue are fed file-transfer and competing traffic at their service rates, the partial flushing queue successfully delivers packets at a rate up to 6% higher. Flushing queues show their real advantage when they are fed data faster than their service rate, though. Such periods could occur frequently if file-transfer applications are allocated less than their peak bandwidth. When a file-transfer application is tuned to network queue sizes properly and when its rate allocation is moderately lower than its peak rate, a partial flush-

ing queue delivers packets at a rate 3.4 times higher than does a FIFO queue (figure 25).



Throughput Ratios for Two Overloaded Queues (Fig. 25)

This figure shows the ratio of delivered packets to offered packets (the throughput ratio) for a partial flushing queue and a FIFO queue fed with combinations of file-transfer data and competing Poisson traffic.

5.5 Staggered Pushout

Video coders often work better with channels that have non-bursty losses than with very bursty losses that occur infrequently. This is first because isolated lost cells usually can be estimated given correct values for the video data in nearby picture regions. For example, a discrete cosine transform (DCT) based video receiver can replace isolated lost brightness coefficients using the median of neighboring brightness coefficients, the past-frame coefficient, or the average of neighboring coefficients. If neighboring brightness coefficients are also lost, estimation performs more poorly. Also, people watching video tend to judge an entire video sequence's quality as equalling the quality during its worst moments. A few serious errors subjectively hurt video quality more than more frequent but less serious errors [32].

Signal processing applications that estimate missing data using neighboring data blocks are served best by transmission channels that never lose more than one cell consecutively. Applications that estimate missing data using more than the nearest neighboring blocks

need transmission channels with even longer intervals between losses, for example once every three, four, or five cells.

A video source could reduce the effective burstiness of cell losses by shuffling its data before passing the data to the network. Data shuffling has several disadvantages, however. Both the video source and receiver need extra memory to store shuffled cells before they are reordered. Thus, both the source and receiver introduce decoding delay into the video link. Also, shuffling must be done over a long enough a window so that a burst of cell losses will not lose all cells in the window. The source has no idea how long burst losses are, however. If burst losses were eliminated by the network, the network itself would know which previous cells have been lost and thus could discard other cells more intelligently.

Next, we present a new buffer access discipline called *staggered pushout*, which reduces the burstiness of cell losses caused by queue overflow compared to the previously described queue access disciplines. Staggered pushout generalizes the priority pushout and FIFO buffer access disciplines. With FIFO, a cell is lost when it arrives at a full queue. With priority pushout, a queued cell is discarded when a higher-priority cell needs its buffer space. With staggered pushout, when a cell arriving at a buffer finds no available buffer space, a cell is chosen to be discarded that is maximally separated from previously discarded cells. That is, the staggered pushout discipline picks a cell to be discarded (either the arriving cell or a queued cell) such that the number of cells successfully delivered between the cell to be discarded currently and previously discarded cells is maximum. Since FIFO, priority pushout, and staggered pushout buffer access disciplines all are work-conserving, all experience the same rate of cell loss when fed with identical inputs. The disciplines differ in their choice of cells to discard.

Through proper choice of cells to be discarded, the cells lost during brief periods of congestion are separated by several successfully delivered cells. Suppose three cells arrive very quickly at a full queue that uses the staggered pushout discipline. The first arriving cell is discarded, since this cell is maximally separated from previous losses. When the second

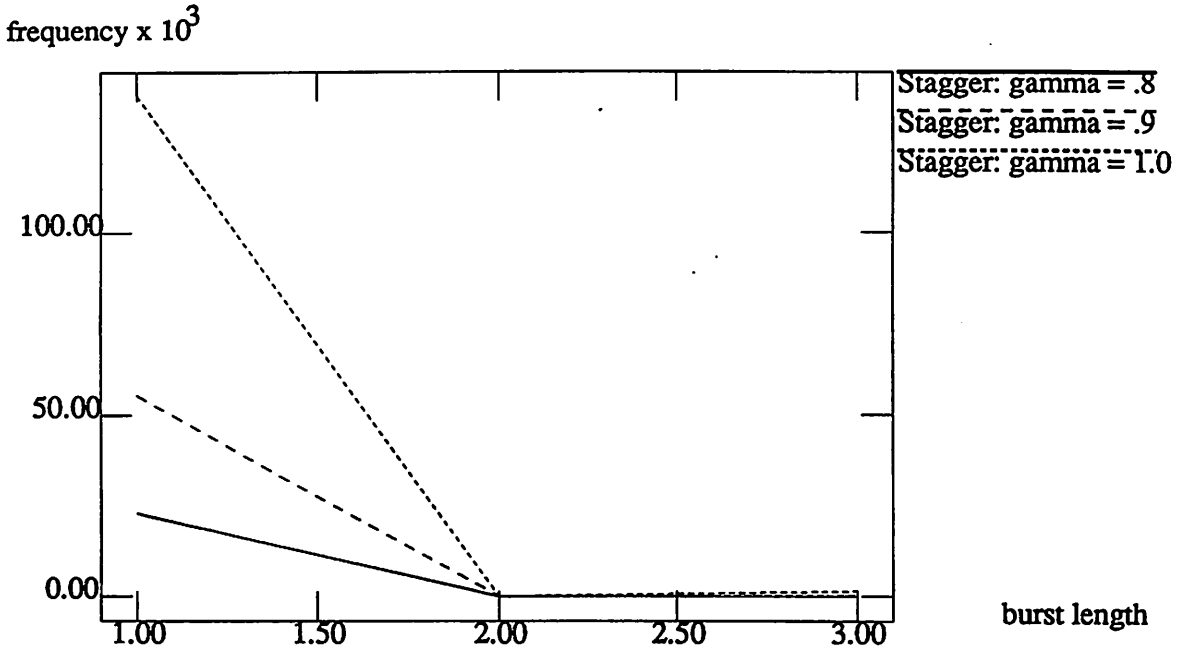
cell arrives, the first (oldest) cell in the queue is discarded (assuming no cells have been lost in a while) to maximize the separation from the first discarded cell. Third, the middle cell in the queue is discarded to maximize the separation from the two previous discards. Rather than losing three consecutive cells, the staggered pushout buffer discards three widely spaced cells.

5.5.1 Simulations

To study analytically the consecutive loss characteristics of a staggered pushout buffer would be very difficult because of the need to track the positions of past losses. Next, we use simulations to show the distributions of the lengths of consecutive cell losses for a FIFO queue and a simplified staggered pushout queue fed by bursty sources. The simplified queue assigns all arriving cells a sequence number. During periods of congestion, this staggered pushout queue bumps out even-numbered cells if any are available; otherwise odd-numbered cells are pushed out. While this simplified discipline does not maximize the separation between cell losses, it does help ensure that no consecutive cells are lost. Even this simplified buffer access discipline allows only a few consecutive cells to be lost in these simulations.

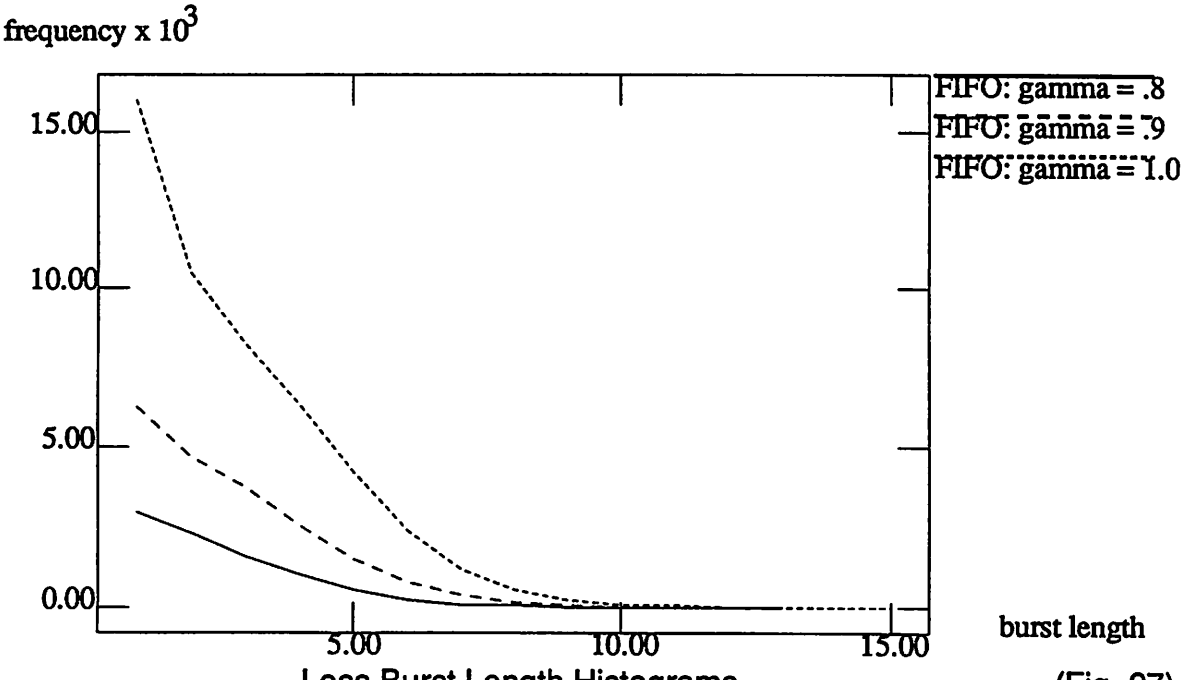
Loss burst length histograms for three different sources are shown in figures 26 and 27. Each source is a two-state Markov-modulated Poisson process (MMPP) [108, 110, 88, 113, 116, 118]. The MMPP generates a Poisson process whose rate depends on the state of a Markov process. The amount of time spent in each state is random and exponentially distributed with a mean time again dependent on the state. The two-state MMPP is characterized with four parameters: for an MMPP(a, b, c, d) process, a is the Poisson rate in state 1, b is the expected holding time of state 1, c is the Poisson rate in state 2, and d is the expected holding time in state 2.

By changing the rates of the two constituent Poisson processes and the expected holding time in each state, simulations of cell arrival streams with varying degrees of burstiness and rate variation can be simulated. Past works generally use Poisson or Bernoulli sources



Loss Burst Length Histograms

(Fig. 26)



Loss Burst Length Histograms

(Fig. 27)

in their simulations; these statistical models cannot adequately model the bursty rate characteristics of real coded video or audio streams, however. Several researchers have used MMPP's to model video with various compression algorithms and source material [88, 72, 76].

In the above simulations each source is modeled as an MMPP(0.84γ, 0.01, 4.2γ, 0.2) process, where γ varies. The simulations run for 3,000,000 iterations (long enough for the queue behaviors to reach steady state) and use an arbitrarily selected queue capacity of 70 cells. For each source (i.e. each γ value) the FIFO and staggered pushout queue lose the same number of cells. However, the staggered pushout queue greatly decreases the length of loss bursts. 66% to 69% of the cell loss bursts in the FIFO queues are longer than one cell long; the bursts are as long as 15 cells. The longest loss burst in the staggered pushout queues is three cells long, and 99% or more of the loss bursts are only a single cell long.

As γ increases, the queues' overall cell loss rates increase and cell loss bursts become longer. Still, the staggered pushout queues limit cell loss bursts to three cells and keep all but 1% of the bursts to only a single cell. With a simple change of buffer access strategy, a network switch can reduce greatly consecutive cell losses in the streams it transports. Alternatively, if a channel must limit the number of consecutive losses it allows, the staggered pushout discipline greatly reduces buffer requirements. Simulations with a FIFO queue with a capacity of 140 cells fed by the above MMPP source with $\gamma = 0.8$ still show up to 6 consecutive cell losses. A queue with the FIFO buffer access discipline would need to be more than twice as large as a queue with the staggered pushout discipline to limit consecutive losses as well.

5.5.2 Video Simulations

We have simulated the transmission of compressed video through both a staggered pushout queue and the partial flushing queue used in section 5.4. The video compression technique used is the same as that used for channel setup negotiations in section 4.5. This technique is similar to that proposed by the MPEG standard [99], but it uses leaky motion compensation to help hide the effects of cell losses [7]. Further, the Huffman coders used by this video coder are modified to allow easy and efficient detection of data losses.

Losses in compressed video streams often occur during scene changes, when interframe compression cannot be used and when bit-rates are higher than average. We simulate the

transmission of a compressed video stream that contains scene changes every fifteen frames, to make cell losses more frequent than they would be with more conventional video input. The compressed stream is fed through the two queues, decoded, and displayed. The stream fed through the staggered pushout queue contains fewer noticeable defects than the stream fed through the partial flushing queue (photos 8 and 9). Although both streams suffer approximately the same overall loss rate, errors are smaller and harder to see in the sequence fed through the staggered pushout queue because the errors are dispersed more widely.

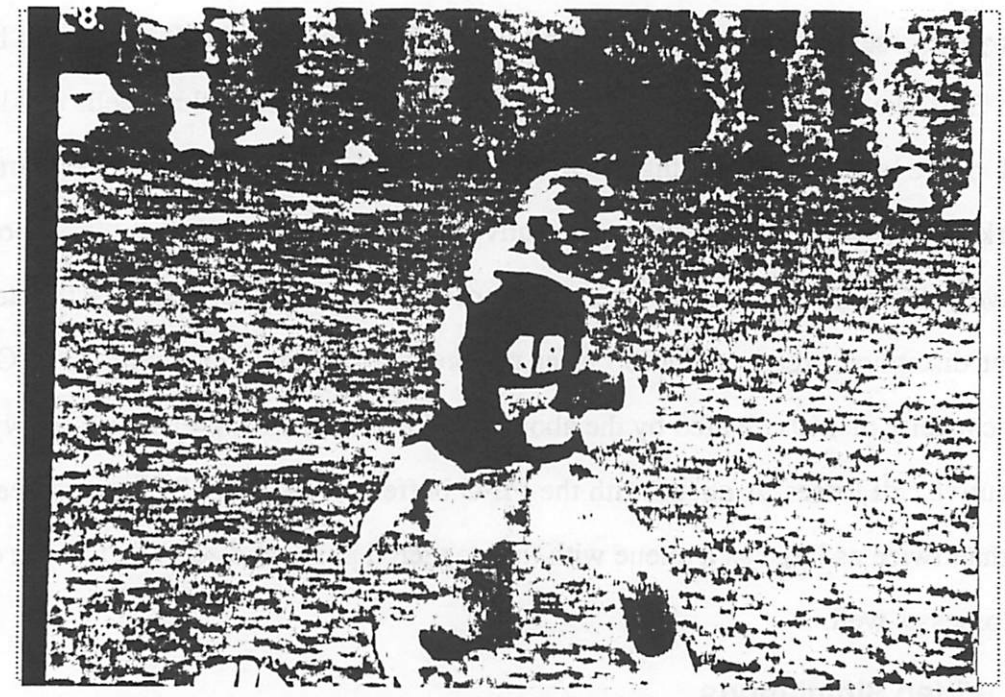


Image from Sequence Transmitted through Staggered Pushout Queue (Photo 8)

5.5.3 Staggered Pushout Rules

Modifications of the staggered pushout discipline certainly are reasonable. Whenever a cell arrives at a full queue, this discipline uses some rule to determine which cell to discard. The choice of a rule produces buffer access disciplines with different properties. The previous section discusses two rules that could be used for a single stream with only one priority class that feeds a single queue. The first rule directs that any cell selected for discard should be maximally separated from previously discarded cells. The simpler rule, suitable

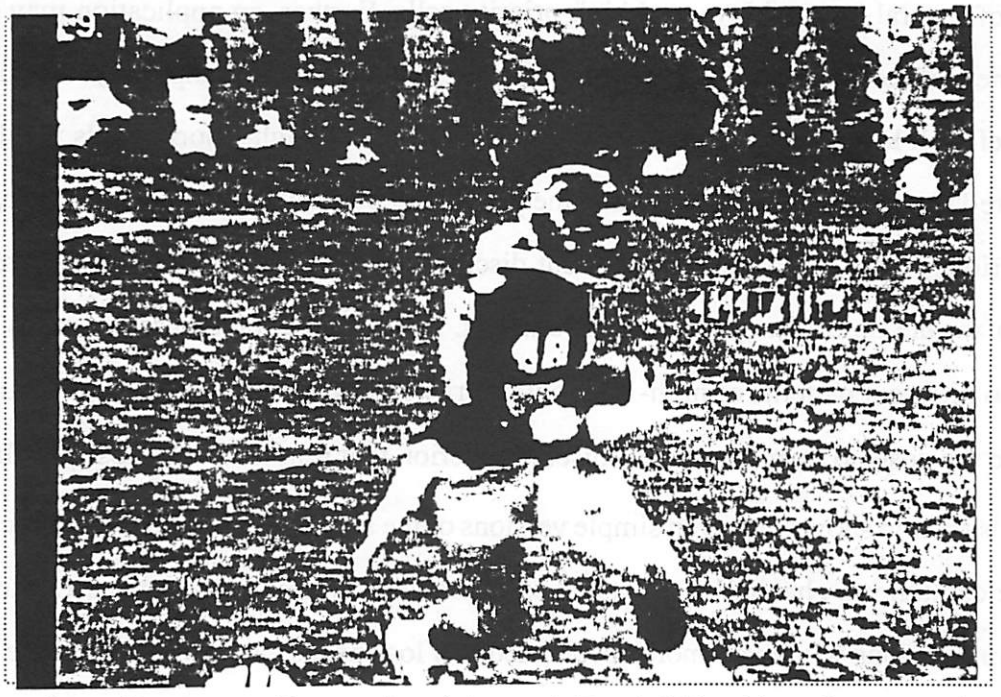


Image from Sequence Transmitted through Partial Flushing Queue (Photo 9)

for applications that estimate lost data using nearest neighbors, directs that a cell to be discarded should not be adjacent to any previously discarded cells. Other rules could follow from other application needs. For example, an application could request that at most two consecutive cells ever be discarded or that at least three cells be delivered successfully between any discarded cells.

All of the above discard rules are examples of "greedy" algorithms—at every time instant these algorithms try to optimize the current loss characteristics without regard to how current decisions may affect future performance. More sophisticated buffer access disciplines could estimate future input characteristics and then act based upon the estimate. For example, if a queue observed that high-speed bursts of inputs were always four cells long, then the queue could discard the cells 0%, 25%, 50%, and 75% of the way down the queue whenever a high-speed burst first is detected.

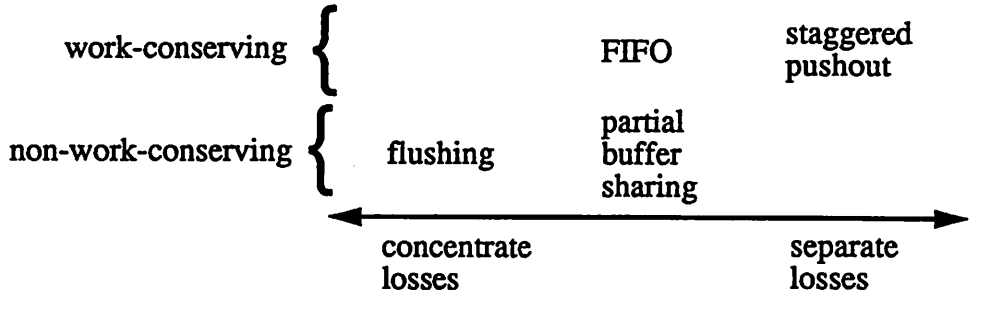
Staggered pushout rules could be generalized to handle multiple priority classes also. With multiple priorities, of course lower priority cells should be discarded before higher-priority cells. However, an application may prefer a long burst of low-priority cell losses to

a mix of several isolated low- and high-priority cells. Further, an application may or may not benefit if lost low- and high-priority cells are widely spaced temporally. Applications that prefer all low-priority cells to be discarded before any high-priority cells would use a staggered pushout discipline much like the prioritized pushout discipline. However, within each priority class, the staggered pushout discipline would discard cells in the order that maximizes the time between consecutive losses. If an application can assign costs or penalties to different patterns of high- and low-priority losses, then a staggered pushout queue can use the costs to formulate a cell-discard criterion that minimizes the loss penalty.

This section shows that even simple versions of the staggered pushout buffer access discipline can tailor a channel's loss statistics to provide short cell loss bursts. Communications applications with even more specialized cell loss needs can have appropriate control methods designed to meet those needs also.

5.6 Comparison of Disciplines

Buffer access disciplines may be classified as work-conserving or non-work-conserving and further as to how much they concentrate or separate cell losses. The disciplines presented in this paper are classified in figure 28.



Buffer Access Discipline Classification (Fig. 28)

A work-conserving loss-concentrating discipline only would discard cells from a full queue and would choose the discarded cells to maximize consecutive loses. We have simulated such a discipline with the file-transfer application discussed in section 5.4 and found that it gives higher packet throughput rates than FIFO but lower than queue flushing. Non-work-conserving queue flushing maintains a longer time between loss bursts than does this

new work-conserving discipline.

Non-work-conserving loss-separating disciplines might be useful in some circumstances. These disciplines would discard cells from partially full queues in hopes of preventing future consecutive losses, trading between low average loss rates and low probabilities of consecutive loss. Many applications would prefer not to suffer higher cell loss rates for only moderate decreases in the rate of consecutive losses, however. These trade-offs could be studied further.

5.7 Hardware Implementation

Above, several experiments have shown that different communications applications, such as video transmission and file transfer, benefit from different network buffer management disciplines that produce channels with loss statistics tuned to the needs of the applications. The simulations implemented buffer management disciplines tuned to the cell loss needs of a single application. A more advanced topic not studied here is the design of buffer management disciplines that simultaneously and efficiently meet the needs of a variety of applications.

We must justify the feasibility of switches that adapt their buffer management disciplines to suit different application requirements. First, simple parameterizable methods such as described in [64] clearly are feasible. [64] describes a technique in which traffic in different priority classes is given different levels of protection against cell discard. A more powerful type of buffer management flexibility would be demonstrated by a switch that routes different types of traffic to different buffers or that selects one of several fixed buffer management disciplines for its internal buffers based upon their traffic mix. Possibly in the future, powerful switches could custom-design in real-time a buffer access discipline tailored for the needs of their current traffic.

Several past works present asynchronous transfer mode (ATM) switch architectures that incorporate flexible processing at the cell level. For example, [47, 70, 82] present switch architectures with dynamic internal routing functions that process each input cell in-

dividually. Similar processing power could be used for buffer management at the cell level. Armbruster [34] analyzes architectural trade-offs in multiuse switches and concludes that switch hardware must be distributed flexibly and extensibly so that it can be deployed dynamically wherever resource needs are greatest. The switch architecture presented in [68] actually includes hardware support for the selection of different buffer management based upon traffic requirements. The architecture combines a non-blocking crossbar stage with ring buffers at each switch output port. A controller attached to each ring buffer decides the order in which stored cells are output. Through the design of clever controllers, it is possible to implement adaptive, modifiable buffer management disciplines within a high-speed switch.

5.8 Conclusion

Different communications applications should use channels with buffer access disciplines tailored to their needs. Applications that transmit data types with varying subjective significance should use a discipline that gives different loss rates to different priority classes. Applications that need a long time between bursts of losses should use a type of queue flushing discipline; for example, partial queue flushing allows designers to trade between the time between loss bursts and the loss probability. Applications that need losses to be spread out should use queues with staggered pushout disciplines. The examples of the file-transfer and the video applications show how the proper choice of buffer access discipline can improve application performance for fixed buffer and bandwidth allocations.

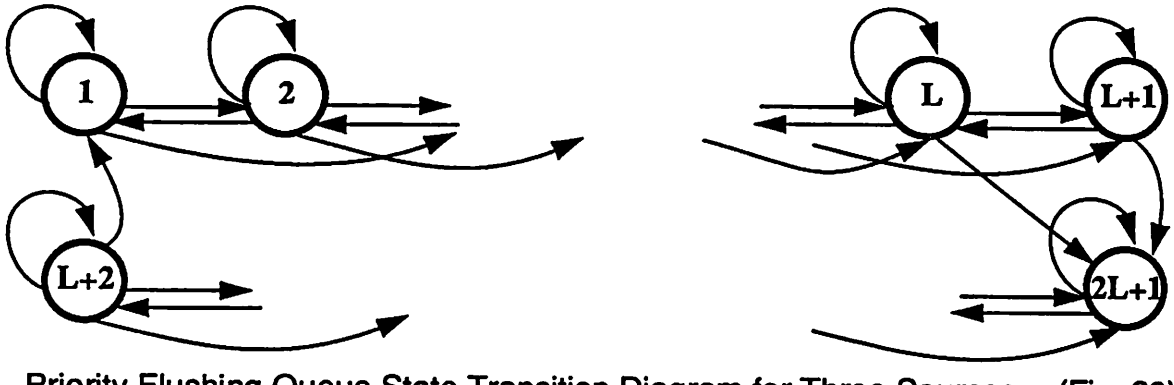
To tailor the buffer access discipline of a channel to meet the QOS needs of its client applications can be much more efficient than to over-allocate resources to a channel to meet those needs with generic buffer management. For example, section 5.5 shows that to guarantee similarly small probabilities that cell loss bursts be no longer than one cell, a FIFO-based queue would need to be larger and served faster than a staggered pushout queue. A network, with knowledge of each of its communication channel's needs, should select its buffer access disciplines so that all applications' quality of service needs are met fairly ex-

actly.

To obtain information on its client applications' needs, a network could use the Medley Interface. Traditional models of the BISDN signaling interface cannot express QOS requests for multiple levels of cell discard priority or for control of a channel's cell loss burstiness. The Medley Interface supports these requests and thus facilitates the use of the signal processing and buffer management disciplines described in this and the preceding chapter. Together, the Medley Interface, adaptable signal processing, and specialized buffer management disciplines allow a variety of communications applications to use networks and network resources more efficiently than do less flexible systems.

5.9 Appendix

To study a prioritized flushing queue of length L, we use a Markov chain with $2L+1$ states. States 1 through $L+1$ represent queue lengths 0 through L when the queue is not flushing low-priority inputs. States $L+2$ through $2L+1$ represent queue lengths 1 through L when the queue is flushing. With k sources, whenever a queue overflow occurs the queue jumps from one of the states $L-k+2$ through $L+1$ directly to state $2L+1$. When the queue is flushing and accepting only high-priority inputs, the Markov chain jumps from state $L+2$ to state 1 as the queue empties. With k sources again and barring overflow, given that the chain is in state x, possible transitions can take place to states $x-1, x, x+1, ..., x+k-1$.



Priority Flushing Queue State Transition Diagram for Three Sources (Fig. 29)

Assume that the priority flushing queue is fed by M high-priority and N low-priority Bernoulli sources, each with probability of arrival equal to p. It is straightforward to find

the transition probabilities for this chain. For example, the probability of moving from state x to state x+1 is the probability of two arrivals in one time period: two arrivals minus one departure equals an increase of one in the queue length. This probability equals $\binom{N+M}{2} p^2 (1-p)^{N+M-2}$.

Given the transition probabilities, we find the Markov chain's state transition matrix Q. From Q we calculate the Markov chain's stationary probability distribution μ. By definition, μ is a normalized vector such that $\mu Q = \mu$. Thus, μ is just the normalized left-eigenvector of Q corresponding to the eigenvalue 1.0, so μ could be found with any eigenvector calculation algorithm such as Jordan decomposition or QR factorization. However, since Q is aperiodic and since all of Q's states are recurrent, we can find μ simply by raising Q to higher and higher powers. Each row of Q^n tends to μ as n increases. From μ we can calculate the percentage of time spent in a flushing state, the percentage of time spent empty, etc.

To calculate the percentage of lost high- and low-priority cells, we condition on the current state. Given that the chain is in state x, we calculate the expected number of cell losses by summing the probability of a quantity of arrivals that will result in cell loss multiplied by the number of losses incurred. In any flushing state, the number of low-priority cells lost equals the number of low-priority cells that arrive.

The priority flushing queue alternates between flushing mode and non-flushing mode. We are interested in the rate r_f at which the queue changes modes. The rate at which the chain leaves flushing mode equals the probability of being in state L+2 times the probability that the transition out of state L+2 is into state 1. The expected time spent in non-flushing mode equals $Pr\{\text{current state is non-flushing}\} / r_f$.

Chapter 6

OVERVIEW OF THE CHANNEL SETUP PROCESS

This thesis has studied three methods for increasing efficiency in high-speed communications applications and modern broadband networks: chapter 3 defined a signaling interface model, chapter 4 discussed video application coding techniques, and chapter 5 presented network buffer management disciplines. Next, an example channel setup negotiation is used to present an overview of all of these techniques and to review how these components cooperate. All three components provide, communicate, or utilize increased knowledge of transport requirements and capabilities to improve application performance and efficiency of network resource use.

Section 4.5 presented channel setup negotiations with a video coder based on motion compensation and the discrete cosine transform (DCT). This chapter reviews the channel establishment process with that coder and discusses how both it and a network would adapt to the negotiation results.

6.1 Preliminaries

Applications and networks both must be designed to support some Medley Interface system requirements in order to participate in channel setup negotiations with this model. The video coder application designer first must decide how to partition its various transmitted data types onto a number of substreams. If each data type is sent over a unique substream with a flowspec tailored to its transport needs, then no data need be sent on a substream with a more expensive rate allocation than necessary. This saves network resources—for example if leaky motion compensation α values and DCT coefficients were both sent over a lossless substream as required for the α values, then the DCT coefficients

would force the allocation of more buffer space than if they were sent separately over a lossy substream.

The video coder in this overview uses three substreams. A "lossless" substream carries very low-rate coder state information such as the motion compensation α value used for each video frame. A second substream carries "high-priority" data whose loss degrades the received image quality significantly, but not as much as the loss of an α value. A third substream carries less important "low-priority" information.

Before channel setup can begin, the video coder and network both must know how to express their transport needs and capabilities. Within the context of the Medley Interface, applications express their transport needs in terms of a *performance function* or *performance level-set* and networks express their capabilities in a *cost function.*

The motion compensation + DCT based video coder performance level-set is designed through a battery of tests and subjective evaluations. The tests simulate video transmission through channels with a range of flowspec parameters—tests show how the video coder should best adapt to different parameters, and they identify parameter sets that yield video of roughly identical subjective quality. The performance level-set identifies these parameters, hiding the details of how the video coder adapts to achieve that performance.

For the video coder in this overview, flowspec parameters that support a relatively constant performance level are listed in table 3 at the end of chapter 4. The "lossless" substream behavior is described with an RLB(*lossless*, 1, 30) rate bound and a LLB(*lossless*, 0, 1) loss bound. The "high-priority" substream's rate is defined with an RLB(*high-priority*, 150, *J*) bound and its loss behavior is specified with an LLB(*high-priority*, 100, 10^7) bound. The "low-priority" substream's behavior is specified with RLB(*low-priority*, 150, *K*), LLB(*low-priority*, *M*, 2*M*), and LLB(*low-priority*, 50, *L*) bounds. All three substreams are given equal delay constraints with the Medley Interface DELAY(·) flowspec bound. The parameters *J*, *K*, *L*, and *M* are established during flowspec parameter negotiations, since the video coder can adapt to operate as these parameters vary over a range.

Similarly, the network decides the resource and processing cost of substreams with a range of flowspec parameters and designs its cost function to reflect the resource costs. The cost function expresses in dollars per minute or some equivalent measure the rate at which an application must pay for its transport service. The cost function hides a channel's implementation details such as the best buffer management discipline for the channel, rate and buffer allocations, etc. In fact the details of a channel's implementation may vary with network conditions at the time of a channel establishment. If a network has allocated relatively little of its buffer space when a channel is established, the channel may be implemented with a lower rate allocation and higher buffer allocation than at other times. This overview uses a substream cost function of the form below; this equation was modeled on the buffer requirements of the file-transfer application discussed in section 3.5.3. The J and K terms charge for reserved bandwidth for the high- and low-priority substreams, the L term charges for reserved buffer space for the low-priority substream, and the M term charges for the control of consecutive losses on the low-priority substream. The constants α, β, κ, λ, and μ are chosen by the network to reflect its relative availability of bandwidth, buffer, and processing resources.

$$cost = \alpha J^{\gamma} + \beta K^{\gamma} + \kappa(\log(L) - \log(6.67)) + \lambda/(M + \mu)$$

6.2 Channel Request

The video application begins to set up a channel by requesting that the network establish a connection to a particular destination. The network then transmits its cost function to the application so that the application can choose minimum-cost channel flowspec parameters that support its desired performance level. The parameters of the cost function are those used by the Medley Interface flowspec description format. At this point the application also must learn about any limitations in network capabilities, such as an inability to base a substream's cost on more than one leaky bucket rate bound.

6.3 Negotiations

The video application uses the iterative minimization algorithm described in section 3.5

with its performance level-set and the cost function it received to find minimum-cost flowspec parameters. This overview reviews the first negotiation presented in section 4.5. The cost function for each substream is:

$$cost = 2J^{0.7} + K^{0.7} + 0.03\,(\log(L) - \log(6.67)) + 0.1/(M+3)$$

The video application picks a reasonable initial set of flowspec parameters: $J = 5.8$ kilocells per second, $K = 5.17$ kilocells per second, $L = 400$ cell deliveries per loss, and $M = 5$ cells. As it runs the iterative minimization algorithm, the application updates its flowspec parameter point in the direction of the projection of the cost function gradient into the local constant-performance plane. The size of the updates varies to keep the updates near the constant-performance plane while attempting to reduce the number of iterations necessary. After 100 iterations the flowspec parameters are $J = 3.15$, $K = 5.60$, $L = 535$, and $M = 4.47$. The channel cost has fallen 21% from 10.1 to 7.96. Thus the network reserves 21% fewer resources with the final parameters than with the initial ones, while the video application maintains the same performance level. The final parameter values are sent to the network in a channel configuration request.

6.4 Network Response

Given the Medley Interface flowspec description of the video application's desired channel, the network can implement the channel. While the application was choosing flowspec parameters, the network could have been calculating prospective routes for the channel. With the flowspec, the network chooses a route that contains sufficient resources to support the channel while minimizing the probability that future channel setup requests will be blocked. The negotiated end-to-end QOS impairments, delay, loss rate, and consecutive losses, must be partitioned among all network components along the channel route. The necessary buffer, bandwidth, and processing resources are reserved along the channel's route, and any switch configuration necessary is performed. For example, since one of the application's substreams has bounded the number of consecutive losses that can occur, the network could use staggered pushout disciplines in its switches to meet this bound with

smaller buffer allocations than would be required with first-in, first-out buffers. Simulations in section 5.5.1 indicate that the buffer savings could be greater than 50%.

Also, the network establishes suitable rate monitors at the application interface and begins to bill the application at the negotiated cost. Possibly, the network also establishes internal monitors that verify that its guaranteed QOS bounds actually are being met.

6.5 Video Coder Response

The coder uses the flowspec parameters that result from iterative cost reduction to adapt its signal processing and multiplexing to these channel characteristics. With the initial channel parameters, the coder would have used a motion compensation α value of 0.875 and 3 high-priority DCT coefficients per block. With the final negotiated flowspec channels, the coder uses $\alpha = 0.75$ and 1 high-priority DCT coefficients per block; these coder parameters are determined by matching the final flowspec parameters with the coder parameters used to generate the known flowspec parameter values in table 3.

After the network confirms to the application that the requested channel was established, the application must tell its receiver how to configure itself. The receiver must know what leaky motion compensation α value to use and how many DCT coefficients per block are sent at high priority. Further, if the multiplexing pattern of each substream's data is not standardized, the application transmitter and receiver must establish a common one. For example, the receiver must know the pattern in which each block's motion vectors and DCT coefficients are sent on the high-priority substream.

The receiver decides, either alone or in conjunction with the transmitter, what corrective actions should be taken when informed of a cell loss. Depending upon the likelihood of cell losses and the availability of processing power, the receiver could use the lost-data estimation techniques described in section 4.4.2 to help reduce the magnitude of image artifacts caused by losses, or it could employ the low-pass filtering concealment techniques of [19] to help make loss-affected video regions less noticeable.

The components of the Medley Interface model enable adaptable video applications and networks to operate more efficiently and with better performance than they could with less flexible interfaces. The Medley Interface substream decomposition and flowspec format enable channel setup negotiations to reduce channel resource consumption, and they give applications and networks sufficient information that they can maximize their performance by adapting to each other's characteristics.

Chapter 7

CONCLUSION

Recent research efforts in the design of both communications networks and applications have led to increased adaptability and flexibility in both domains, yielding several advantages. Flexible networks support a larger variety of applications and facilitate the introduction of new applications because they already provide high-level transport functions such as sequenced delivery and loss-detection; each new application need not implement these functions. Flexible applications will be successful because they can operate with a wide variety of networks without modification. As wireless networks, BISDN's, and high-speed local area networks all are deployed to meet different communications needs, flexible applications that operate seamlessly with all available networks will be in demand.

New network buffer management disciplines allow networks to transport diverse traffic mixes quickly and efficiently [35, 45, 53, 64, 75, 85]. Network interface models such as those developed by Washington University [39, 41] and Bellcore [36, 37, 51, 58, 66, 74] allow applications to establish multiway connections in a simple, intuitive way.

Communications applications such as video transmission have become more adaptable also. Recent coding formats such as JPEG, MPEG, and the digital U. S. high-definition television proposals [97, 92, 94, 96, 98, 99, 103] all specify several different video compression techniques that can be chosen to adapt to different source material or performance demands. These formats also are *scalable*, which means they can be applied to a range of picture sizes and frame rates.

An obstacle to the utilization of increasing flexibility in networks and applications is the interface between them. Many currently proposed channel establishment interfaces are poorly defined or are so simple as to prevent efficient network operation. A more powerful

model is needed to support guaranteed transport quality of service (QOS) to applications and to enable efficient processing and resource use within networks.

This thesis has studied three methods for increasing efficiency in high-speed communications applications and modern broadband networks: a signaling interface model, video application coding techniques, and network buffer management disciplines. Chapter 3 presented a new model for the channel setup interface between applications and networks. The Medley Interface model allows applications to specify their transport QOS needs in more detail than current BISDN models. An application can request data rate, delay, loss rate, and loss burstiness or spacing guarantees (i. e. flow specification or *flowspec* guarantees) on several substreams that combine to form a single channel. Substream decomposition combined with the Medley Interface's detailed flowspec format allows applications designers to specify QOS guarantees in more detail and to design applications that can operate with a wider variety of available channels than is now possible. These features of the Medley Interface also allow networks to implement channels using link and buffer resources more economically and to tailor channels' characteristics more closely to application needs. For example, a network channel for the video coder described in chapter 4 could use up to 27% less buffer space if it prevents consecutive cell losses.

This interface model also allows more powerful call setup negotiations than previously studied. The negotiations benefit from the wide range of trade-offs between channel characteristics that can be expressed with the detailed flowspec format and substream decomposition described above. Section 3.5.2 has presented an iterative gradient-descent minimization algorithm that, by exchanging cost gradient information as well as cost data, allows applications and networks to obtain minimum-cost channels that support the applications' needs rather than simply to establish and use the first feasible channels that the applications request. Prototype negotiations with a file-transfer and conditional-replenishment video application have achieved significant channel cost reductions.

Chapter 4 next reviewed two modern video compression techniques, motion compen-

sation and the discrete cosine transform (DCT) and has presented new modifications of the standard motion compensation algorithm that improve its resilience to transmission losses. Leaky motion compensation, conditional-replenishment, and conditional leaky motion compensation all produce video with higher subjective quality in the presence of cell losses than does motion compensation with standard periodic error replenishment. These new coding techniques allow video coders based on motion compensation and the DCT to provide high-quality video with a range of channel QOS parameters, and thus they allow such coders to negotiate for low-cost channels with Medley Interface networks. During simulated negotiations, the video application has maintained a fixed video performance level while reducing its channel cost by up to 70%.

Finally chapter 5 presented new buffer management disciplines that allow network switches to efficiently provide communications channels with the loss-spacing characteristics desired by a range of applications. Applications such as file-transfer that must reset their transmitter and receiver after every cell loss operate most efficiently if losses are tightly bunched and groups of losses are widely spaced. A queue flushing, partial queue flushing, or priority queue flushing discipline provides these loss characteristics and gives file-transfer applications packet throughput rates as much as 3.4 times than a first-in, first-out (FIFO) queue.

Applications such as video transmission that can operate in spite of cell losses operate best if loses are separated by as many successfully delivered cells as possible. Widely spaced losses subjectively are less objectionable, and they allow lost data estimation to perform more reliably than consecutive losses. The staggered pushout discipline gives channels widely spaced cell losses and thus supports video applications more efficiently than FIFO or flushing disciplines. Simulations in section 5.5.1 showed that staggered pushout discipline queues with 50% the capacity of FIFO queues still limit consecutive cell losses more effectively.

Together, the signaling interface model, video coder adaptations, and buffer manage-

ment disciplines presented in this thesis show the benefits and feasibility of a richer call set-up interface for the BISDN than has been envisioned. Video applications can operate with a range of channel QOS parameters, but they must have some control of the parameters to produce video with high subjective quality after transmission. Networks can provide channels with delay, loss rate, loss priority, and loss spacing characteristics finely tuned to the needs of specific applications, but these needs must be made known to the network. These channel characteristics are specified with the Medley Interface flowspec format; this format further facilitates channel setup negotiations that minimize (with some limitations) an application's transmission cost at a fixed performance level.

7.1 Future Work

Many open problems remain in the study of interactions between networks and their client applications. Progress with the network resource allocation problem is proceeding slowly. Although researchers have derived analytic expressions for the bandwidth and storage requirements for simple traffic models and QOS parameters, the extension of this work to more realistic models or to mixes of several traffic types has not been very successful. Resource allocation heuristics based on network simulations yield useful results, but they are difficult to generalize to other traffic mixes. In fact, the composition of future BISDN traffic is largely unknown!

The establishment of pricing strategies for BISDN's is related to the resource allocation problem. Good pricing strategies discourage the waste of network resources and processing power. The price of a channel should be "fair" in that it is somewhat proportional to the network effort expended to implement the channel. The design of effective pricing strategies is largely unexplored.

The design of high-speed network hardware architectures that can adapt to a variety of application needs is in its infancy. Although researchers have presented architectures that vary their buffer management or routing depending on short-term traffic characteristics [47, 68, 70, 82], more work must be done to increase switches' flexibility at high data rates.

The next few years surely will see advances in signal processing and estimation methods to hide the effects of data loss and delay. As BISDN's, wireless networks, and local area networks become faster and more prevalent, they will carry a growing amount of video traffic. To operate efficiently, applications that use any of these networks must tolerate somewhat bursty data losses.

The channel setup negotiations presented in this thesis show the feasibility of cost-minimization negotiations, but the method employed in this thesis is inadequate in some ways. Any strict gradient-descent method becomes trapped in local minima of the cost function too easily. Also, more study is needed to design application performance functions more efficiently and exactly and to approximate them more accurately during negotiations.

Negotiations with subjectively-defined performance level-sets occasionally terminate because iterations carry the current flowspec parameter point to the boundary of the defined performance level-set. Negotiations could achieve significantly greater cost reductions if they could continue along the boundary rather than terminating. In general, the development of fast, efficient channel setup negotiations that require low communications overhead should see attention in the future.

Chapter 8

REFERENCES

8.1 Video Coding for Cell Relay Networks

[1] J. Darragh and R. Baker, "Fixed Distortion Subband Coding of Images for Packet-Switched Networks," *IEEE Journal on Selected Areas in Communications*, vol. 7, June 1989, pp. 789-800.

[2] M. Garrett and M. Vetterli, "Congestion Control Strategies for Packet Video," *International Packet Video Workshop*, Kyoto, August 1991, pp. G2.1-G2.6.

[3] M. Ghanbari, "Two-Layer Coding of Video Signals for VBR Networks," *IEEE Journal on Selected Areas in Communications*, vol. 7, June 1989, pp. 771-781.

[4] M. Ghanbari and J. Azari, "Comparison between the CBR and VBR at the base layer of a Two-layer video codec," *International Packet Video Workshop*, Kyoto, August 1991, pp. F2.1-F2.6.

[5] P. Haskell, K. Tzou, and T. Hsing, "A Lapped-Orthogonal-Transform Based Variable Bit-Rate Video Coder for Packet Networks," *IEEE ICASSP'89*, May 1989, Glasgow, Scotland, pp. 1905-1908.

[6] P. Haskell and D. Messerschmitt, "Reconstructing Lost Video Data in a Lapped Orthogonal Transform Based Coder," *IEEE ICASSP'90*, April 1990, Albuquerque, New Mexico, pp. 1985-1988.

[7] P. Haskell and D. Messerschmitt, "Resynchronization of Motion Compensated Video Affected by ATM Cell Loss," *IEEE ICASSP'92*, March 1992, San Francisco, California.

[8] G. Karlsson and M. Vetterli, "Sub-band Coding of Video Signals for Packet-Switched Networks," *SPIE Visual Communications and Image Processing II*, vol. 845, 1987, pp. 446-456.

[9] G. Karlsson and M. Vetterli, "Three Dimensional Sub-Band Coding of Video," *IEEE ICASSP*, 1988, pp. 1100-1103.

[10] G. Karlsson and M. Vetterli, "Packet Video and its Integration into the Network Architecture," *IEEE Journal on Selected Areas in Communications*, vol. 7, June 1989, pp. 739-751.

[11] F. Kishino, et. al., "Variable Bit-Rate Coding of Video Signals for ATM Networks," *IEEE Journal on Selected Areas in Communications*, vol. 7, June 1989, pp. 801-806.

[12] D. Lee, S. Li, and K. Tzou, "Analysis of Video Packet Loss Control in ATM Networks," *IEEE Globecom*, 1990, pp. 857-861.

[13] D. Lee, K. Tzou, and S. Li, "Control Analysis of Video Packet Loss in ATM Networks," *SPIE Visual Communications and Image Processing*, vol. 1360, 1990, pp. 1232-1242.

[14] J. Ohm, "A Hybrid Image Coding Scheme for ATM Networks Based on SBC-VQ and Tree Encoding," *International Workshop on Packet Video*, August 1991, Kyoto Japan, pp. B2.1-B2.6.

[15] A. Puri and R. Aravind, "An Interframe Coding Scheme for Packet Video," *SPIE Visual Communications and Image Processing IV*, vol. 1199, 1989, pp. 1610-1618.

[16] A. Riebman, P. Goli, and V. Kumar, "Combined Performance Study of a Video Compression System and an ATM Switch," *International Packet Video Workshop*, Kyoto, August 1991, pp. E4.1-E4.6.

[17] K. Sakai, T. Hamano, and K. Matsuda, "Varible Bit-Rate Video Coding with Cell-Loss Compensation," *International Packet Video Workshop*, Kyoto, August 1991, pp. C10.1-C10-6.

[18] M. Tsujikado, et. al., "A Variable Bit Rate Video Coding Scheme for Broadcast Quality Service via ATM Networks," *International Packet Video Workshop*, Kyoto, August 1991, pp. C11.1-C11.6.

[19] K. Tzou, "Post Filtering for Cell Loss Concealment in Packet Video," *SPIE Visual Communications and Image Processing IV*, vol. 1199, 1989, pp. 1620-1628.

[20] W. Verbiest, "Variable Bit Rate Video Coding in an ATD Network," *Proceeding of the Picture Coding Symposium*, June 1987, Stockholm, Sweden, pp. 200-201.

[21] W. Verbiest, L. Pinnoo, and B. Voten, "The Impact of the ATM Concept on Video Coding," *IEEE Journal on Selected Areas in Communications*, vol. 6, December 1988, pp. 1623-1632.

[22] W. Verbiest and L. Pinnoo, "A Variable Bit Rate Video Codec for Asynchronous Transfer Mode Networks," *IEEE Journal on Selected Areas in Communications*, vol. 7, June 1989, pp. 761-770.

8.2 General Video Coding

[23] W. Chen and W. Pratt, "Scene Adaptive Coder," *IEEE Transactions on Communications*, vol. COM-32, March 1984, pp. 225-232.

[24] D. Connor, "Techniques for Reducing the Visibility of Transmission Errors in Digitally Encoded Video Signals," *IEEE Transactions on Communications*, June 1973, vol. 21:3, pp. 695-706.

[25] M. Gilge, T. Engelhardt, and R. Mehlan, "Coding of Arbitrarily Shaped Image Seg-

ments Based on a Generalized Orthogonal Transform," *Signal Processing: Image Communication*, vol. 1:2, October 1989, pp. 153-180.

[26] T. Hsing, ""Motion Detectors and Compensation Coding for Motion Video Coders: Technical Review and Comparison," *IEEE Globecom*, 1987, pp. 2.6.1-2.6.5.

[27] A. Jain, "A Sinusoidal Family of Unitary Transforms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, October 1979, pp. 356-365.

[28] A. Jain, *Fundamentals of Digital Image Processing*, Englewood Cliffs, NJ, Prentice Hall, 1989.

[29] A. Netravali and B. Haskell, *Digital Pictures*, New York, Plenum Press, 1988.

[30] A. Netravali and J. Robbins, "Motion-Compensated Television Coding: Part I," *Bell System Technical Journal*, vol. 58, March 1979, pp. 631-670.

[31] D. Taubman and A. Zakhor, "A Multi-Start Algorithm for Signal Adaptive Subband Systems," *IEEE ICASSP*, March 1992, pp. III.213-III.216.

[32] S. Wolf, "The Development and Evaluation of an Objective Video Quality Assessment System that Emulates Human Viewing Panels," *AT&T Workshop on Quality of Service Issues in High Speed Networks*, April 1992, Murray Hill, NJ.

8.3 Cell Relay Networks

[33] V. Anantharam, "The Optimal Buffer Allocation Problem," *IEEE Transactions on Information Theory*, July 1989, pp. 721-725.

[34] B. Armbruster and F. Mellor, "Switch Architecture Evolution in SONET Networks," *IEEE International Conference on Communications*, Atlanta, GA, April 1990, pp. 552-556.

[35] G. Awater and F. Schoute, "Optimal Queueing Policies for Fast Packet Switching of Mixed Traffic," *IEEE Journal on Selected Areas in Communications*, April 1991, vol. 9:3, pp. 458-467.

[36] M. Bahl, J. Daane, and R. O'Grady, "The Evolving Intelligent Interexchange Network - An SS7 Perspective," *Proceedings of the IEEE*, April 1992, vol. 80:4, pp. 637-643.

[37] R. Berman, "Perspectives on the AIN Architecture," *IEEE Communications Magazine*, Feb 1992, vol. 30:2.

[38] D. Bertsekas and R. Gallager, *Data Networks*, 2nd ed., Englewood Cliffs, NJ, Prentice Hall, 1992.

[39] R. Bubenik, J. DeHart, and M. Gaddis, "Multipoint Connection Management in High Speed Networks," *IEEE INFOCOM*, 1991, vol. 1, pp. 59-68.

[40] M. Butto, E. Cavallero, and A. Tonietti, "Effectiveness of the 'Leaky Bucket' Polic-

ing Mechanism in ATM Networks," *IEEE Journal of Selected Areas in Communications*, April 1991, vol. 9:3, pp. 335-342.

[41] J. Cox, M. Gaddis, and J. Turner, "Project Zeus," *IEEE Network*, March 1993, vol.7:2, pp. 20-30.

[42] T. Cox, et. al., "SMDS: The Beginning of WAN Superhighways," *Data Communications*, vol. 20, April 1991, pp. 105-110.

[43] M. de Prycker, *Asynchronous Transfer Mode: Solution for Broadband ISDN*, New York, Ellis Horwood, 1991.

[44] M. Decina, L. Faglia, and T. Toniatti, "Bandwidth Allocation and Selective Discarding for Variable Bit Rate Video and Bursty Data Calls in ATM Networks," *IEEE INFOCOM*, April 1991, vol. 3, pp. 1386-1393.

[45] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," *SIGCOMM'90*, Sept. 1989, Austin, TX, *Computer Communications Review*, vol. 19:4.

[46] A. Elwalid and D. Mitra, "Stochastic Fluid Models in the Analysis of Access Regulation in High Speed Networks," *IEEE Globecom'91*, pp. 1626-1637.

[47] K. Eng, M. Karol, and Y.-S. Yeh, "A Growable Packet (Atm) Switch Architecture: Design Principles and Application," *IEEE Transactions on Communications*, Feb. 1992, vol.40:2, pp. 423-430.

[48] D. Ferrari, A. Banerjea, and H. Zhang, "Network Support for Multimedia: A Discussion of the Tenet Approach," *International Computer Science Institute Technical Report*, TR-92-072, Berkeley, CA, November 1992.

[49] D. Ferrari and D. Verma, "A Scheme for Real-Time Channel Establishment in Wide-Area Networks," *IEEE Journal on Selected Areas in Communications*, April 1990, vol, 8:3, pp. 368-379.

[50] A. Fraser, "Design Considerations for a Public Data Network," *IEEE Communications Magazine*, October 1991, pp. 31-35.

[51] M. Fujioka, H. Yagi, and Y. Ikeda, "Universal Service Creation and Provision Environment for Intellignet Networks," *IEEE Communications Magazine*, Jan 1991, vol. 29:1, pp. 44-51.

[52] R. Gibbens and P. Hunt, "Effective Bandwidths For The Multitype UAS Channel," *Queueing Systems*, October 1991, vol. 9:1-2, pp. 17-27.

[53] S. Golestani, "Congestion-Free Communication in High-Speed Packet Networks," *IEEE Transactions on Communications*, Dec 1991, vol. 39, #12, pp. 1802-1812.

[54] R. Guerin, H. Ahmadi, and M. Naghshineh, "Equivalent Capacity and its Application to Bandwidth Allocation in High-Speed Networks," *IEEE Journal on Selected*

Areas in Communications, Sept. 1991, vol.9:7, pp. 968-981.

[55] D. Hong and T. Suda, "Congestion Control and Prevention in ATM Networks," *IEEE Network*, July 1991, vol.5:4, pp. 10-16.

[56] J. Hui, "Resource Allocation for Broadband Networks," *IEEE Journal on Selected Areas in Communications*, vol. 6, December 1988, pp. 1598-1608.

[57] J. Hui, *Switching and Traffic Theory for Integrated Broadband Networks*, Kluwer Academic Publishers, 1991.

[58] "Intelligent Networks," *IEEE Communications Magazine*, Feb 1992, vol. 30:2.

[59] S. Jordan and P. Varaiya, "Throughput in Multiple Service, Multiple Resource Communication Networks," *IEEE Transactions on Communications*, vol. 39, August 1991, pp. 1216-1222.

[60] C. Kalmanek, H. Kanakia, and S. Keshav, "Rate Controlled Servers for Very High-Speed Networks," *IEEE Globecom*, vol. 1, 1990, pp. 12-20.

[61] G. Kaplan, "Data Communications," *IEEE Spectrum*, vol. 28, August 1991, pp. 21-45.

[62] F. Kelly, "Effective Bandwidths At Multiclass Queues," *Queueing Systems*, October 1991, vol. 9:1-2, pp. 5-15.

[63] M. Kiemele, "Optimizing the Topology of an Integrated Network," in *Network Modeling, Simulation, and Analysis*, edited by R. Garzia and M. Garzia, New York, Marcel Dekker, 1990, pp. 329-359.

[64] A. Lin and J. Silvester, "Priority Queueing Strategies and Buffer Allocation Protocols for Traffic Control at an ATM Integrated Broadband Switching System," *IEEE Journal on Selected Areas in Communications*, Dec 1991, vol. 9:9, pp. 1524-1536.

[65] J. McQuillan, "Broadband Networks," *Data Communications*, vol. 19, June 1990, pp. 76-86.

[66] S. Minzer, "A Signaling Protocol for Complex Multimedia Services," *IEEE Journal on Selected Areas in Communications*, Dec. 1991, vol.9:9, pp. 1383-1394.

[67] S. Minzer, "Broadband ISDN and Asynchronous Transfer Mode (ATM)," *IEEE Communications Magazine*, vol. 27, September 1989, pp. 17-24, 57.

[68] B. Monderer, G. Pacifici, and C. Zukowski, "The Cylinder Switch: an Architecture for a Manageable VLSI Giga-cell Switch," *IEEE International Conference on Communications*, Atlanta, GA, April 1990, pp. 567-571.

[69] A. Netravali, W. Roome, and K. Sabnani, "Design and Implementation of a High-speed Transport Protocol," *IEEE Transactions on Communications*, Nov. 1990, vol.38:11, pp. 2010-2024.

[70] H. Obara, "Distributed ATM Cross-connect Switch Architecture Using Transmission Scheduling Control," *Electronics and Communications in Japan, Part 1 (Communications)*, Jan. 1991, vol.74:1, pp. 55-64.

[71] H. Ohta and T. Kitami, "A Cell Loss Recovery Method Using FEC in ATM Networks,"*IEEE Journal on Selected Areas in Communications*, Dec 1991, vol. 9:9, pp. 1471-1483.

[72] E. Rathgeb, "Modeling and Performance Comparison of Policing Mechanisms for ATM Networks," *IEEE Journal on Selected Areas in Communications*, April 1991, pp. 325-334.

[73] J. Roberts, "Variable-Bit-Rate Traffic Control in B-ISDN," *IEEE Communications Magazine*, September 1991, pp. 50-56.

[74] R. Robrock, "The Intelligent Network - Changing the Face of Telecommunications," *Proceedings of the IEEE*, Jan 1991, vol. 79:1, pp. 7-20.

[75] H. Saito, "Optimal Queueing Discipline for Real-Time Traffic at ATM Switching Nodes," *IEEE Transactions on Communications*, December 1990, vol. 38:12, pp. 2131-2136.

[76] H. Saito, M. Kawarasaki, and H. Yamada, "An Analysis of Statistical Multiplexing in an ATM Transport Network," *IEEE Journal on Selected Areas in Communications*, vol. 9, April 1991, pp. 359-367.

[77] R. Sharma, *Network Topology Optimization: The Art and Science of Network Design*, New York, Van Nostrand Reinhold, 1990.

[78] "Special Issue on ISDN," *IEEE Communications Magazine*, vol. 28, April 1990.

[79] J. Suruagy Monteiro, M. Gerla, and L. Fratta, "Input Rate Control for ATM Networks," Queueing, Performance and Control in ATM: Proceedings of the Thirteenth International Teletraffic Congress Copenhagen, Denmark, June 1991, pp. 117-122.

[80] C. Topolcic, *Experimental Internet Stream Protocol, Version 2 (ST-II)*, Internet RFC 1190, October 1990.

[81] D. Towsely and S. Tripathi, "A Single Server Priority Queue with Server Failures and Queue Flushing," Operations Research Letters, Aug. 1991, vol. 10:6, pp. 353-362.

[82] T. Troudet and S. Walters, "Neural Network Architecture For Crossbar Switch Control," *IEEE Transactions on Circuits and Systems*, Jan. 1991, vol.38:1, pp. 42-56.

[83] J. van der Rhee and F. Schoute, "ATM Traffic Capacity Modelling," *Philips Telecommunication and Data Systems Review*, June 1990, vol.48:2, pp. 24-32.

[84] H. Zhang and D. Ferrari, "Rate-Controlled Static-Priority Queueing," *IEEE Infocom'93*, 1993, San Francisco, pp. 227-236.

[85] H. Zhang and S. Keshav, "Comparison of Rate-Based Service Disciplines," ACM SIGCOMM'91 Conference, vol. 21:4, September 1991, pp. 113-121.

[86] L. Zhang, et. al., "RSVP: A New Resource ReSerVation Protocol," preliminary draft, to appear in *Proceedings of ACM SIGCOMM*, 1993.

8.4 Rate Characteristics of Coded Video

[87] S. Huang, "Source Modelling for Packet Video," *IEEE International Conference on Communications*, 1988, pp. 1262-1267.

[88] D. Lee and S. Li, "Transient Analysis of Multi-Server Queues with Markov-Modulated Poisson Arrivals and Overload Control," *Performance Evaluation*, vol.16, #1-3, Nov. 1992, pp. 49-66.

[89] B. Maglaris, et. al., "Performance Models of Statistical Multiplexing in Packet Video Communications," *IEEE Transactions on Communications*, vol. 36, July 1988, pp. 834-843.

[90] M. Nomura, T. Fujii, and N. Ohta, "Basic Characteristics of Variable Rate Video Coding in ATM Environment," *IEEE Journal on Selected Areas in Communications*, vol. 7, June 1989, pp. 752-760.

[91] M. Zukerman, M. Leditschke, and M. Biggar, "Traffic Studies of Variable Bit Rate Conferencing Video," *International Packet Video Workshop*, Kyoto, August 1991, pp. D2.1-D2.6.

8.5 Proposals and Standards

[92] *Advanced Digital Television System Description*, proposal submitted to the Federal Communications Commission by the Advanced Television Research Consortium, 27 February 1991.

[93] P. Ang, P. Ruetz, and D. Auld, "Video Compression Makes Big Gains," *IEEE Spectrum*, vol. 28, October 1991, pp. 16-19.

[94] *ATVA-Progressive System*, proposal submitted to the Federal Communications Commission by the American Television Alliance, February 1991.

[95] "Coding of Moving Pictures and Associated Audio," MPEG Video Committee ISO-IEC/JTC1/SC2/WG11 Draft of Standard ISO 11172, 18 December 1990.

[96] *DigiCipher HDTV System*, proposal submitted to the Federal Communications Commission by General Instrument Corporation, 6262 Lusk Blvd., San Diego, CA, 92121, 8 June 1990.

[97] "Digital Compression and Coding of Continuous-Tone Still Images," JPEG Committee ISO-IEC/JTC1 Draft ISO 10918, 1991.

[98] *Digital Spectrum Compatible*, proposal submitted to the Federal Communications

Commission by American Telephone and Telegraph (AT&T) and Zenith Electronics Corporation.

[99] D. LeGall, "MPEG: A Video Compression Standard for Multimedia Applications," *Communications of the ACM*, vol. 34, April 1991, pp. 47-58.

[100] M. Liou, "Overview of the p x 64 kbps Video Coding Standard," *Communications of the ACM*, vol. 34, April 1991, pp. 59-63.

[101] B. Markey, "HyTime and MHEG," *IEEE CompCon Proceedings*, February 1992, pp. 25-40.

[102] "Video Codec for Audiovisual Services at p x 64 kbits," International Telephone and Telegraph Consultative Committee (CCITT) *Recommendation H.261*.

[103] G. Wallace, "The JPEG Still Picture Compression Standard," *Communications of the ACM*, vol. 34, April 1991, pp. 31-44.

[104] "Asynchronous Transfer Mode (ATM) and ATM Adaptation Layer (AAL) Protocols Generic Requirements," Bellcore Technical Advisory TA-NWT-001113, August 1992.

8.6 Other

[105] N. Behki and S. Tavares, "An Integrated Approach to Protocol Design," *IEEE Pacific Rim Conference on Communications*, Computers and Signal Processing, June 1989, pp. 244-248.

[106] J. Buck, et. al., "The Almagest: Manual for Ptolemy", University of California, Berkeley Electronics Research Labs report.

[107] T. Cover and J. Thomas, *Elements of Information Theory*, New York, John Wiley and Sons, Inc., 1991.

[108] W. Ding, "Calculating the Parameters of a Two-Phase Markov-Modulated Poisson Process Using Moments of the Interarrival Time," *ITG-Fachberichte*, vol. 107, 1989, pp. 125-130.

[109] G. Golub and C. Van Loan, *Matrix Computations*, Baltimore, The Johns Hopkins University Press, 1989.

[110] H. Heffes, "A Class of Data Traffic Processes-Covariance Function Characterization and Related Queueing Results," *Bell Sys. Tech. Journal*, vol. 59:6, 1980, pp. 897-929.

[111] G. Holzmann, "Design and Validation of Protocols: a Tutorial," *Computer Networks and ISDN Systems*, April 1993, vol. 25:9, pp. 981-1017.

[112] E. Lee and D. Messerschmitt, *Digital Communication*, Boston, Kluwer Academic Publishers, 1988.

[113] M. Neuts, "A Versatile Markovian Point Process," *Journal of Applied Probability*, vol. 16, 1979, pp. 764-779.

[114] R. Otten and L. van Ginneken, *The Annealing Algorithm*, Boston, Kluwer Academic Publishers, 1989.

[115] S. Sheng, A. Chandrakasan, and R. Broderson, "A Portable Multimedia Terminal," *IEEE Communications Magazine*, vol. 30:12, December 1992, pp. 64-75.

[116] H. Sitaraman, "Approximation of some Markov-Modulated Poisson Processes," *ORSA Journal on Computing*, vol. 3:1, Winter 1991, pp. 12-22.

[117] M. Stonebraker, "An Overview of the Sequoia 2000 Project," *COMPCON'92*, February 1992, pp. 383-388.

[118] U. Sumita and Y. Masuda, "Analysis of Multivariate Markov Modulated Poisson Processes," *Operations Research Letters*, vol. 12:1, July 1992, pp. 37-45.

[119] M. van Sinderen, L. Ferreira Pires, and C. Vissers, "Protocol Design And Implementation Using Formal Methods," *Computer Journal*, Oct. 1992, vol.35:5, pp. 478-491.