https://ntrs.nasa.gov/search.jsp?R=19980007076 2020-06-16T00:39:23+00:00Z

Final Technical Report, Volume 1 • February 1994

# REGULAR TOPOLOGIES FOR GIGABIT WIDE-AREA NETWORKS

ITAD-8600-FR-94-005

Volume 1

Project 8600

Prepared by:

Nachum Shacham, Program Director
Barbara A. Denny, Computer Scientist
Diane S. Lee, Senior Research Engineer
Irfan H. Khan, Research Engineer
Danny Y.C. Lee, Research Engineer
Paul McKenney
Telecommunications Theory and Technology Program

Prepared for:

National Aeronautics and Space Administration
Ames Research Center
Moffett Field, California 94035

Attn: Dr. Henry Lum, Code RI, M/S: 244-7

and

Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, Virginia 22203-1714

Attn: Major Michael St. Johns

Approved by:

Michael S. Frankel, Vice President and Director
Information, Telecommunications, and Automation Division

SRI International 333 Ravenswood Avenue • Menlo Park, CA 94025-3493 • (415) 326-6200 • FAX: (415) 326-5512 • Telex: 334486

# CONTENTS

**FIGURES**

# 1 INTRODUCTION AND OVERVIEW

SRI International (SRI) is pleased to submit this final report describing our efforts under SRI Project 8600, entitled "Regular Topologies for Gigabit Wide-Area Networks." In general terms, this project aimed at the analysis and design of techniques for very high-speed networking. The formal objectives of the project were to

- Identify switch and network technologies for wide-area networks that interconnect a large number of users and can provide individual data paths at gigabit/s rates
- Quantitatively evaluate and compare existing and proposed architectures and protocols, identify their strength and growth potentials, and ascertain the compatibility of competing technologies
- Propose new approaches to existing architectures and protocols, and identify opportunities for research to overcome deficiencies and enhance performance.

The project was organized into two parts:

1. The design, analysis, and specification of techniques and protocols for very-high-speed network environments. In this part, SRI has focused on several key high-speed networking areas, including forward error control (FEC) for high-speed networks in which data distortion is the result of packet loss, and the distribution of broadband, real-time traffic in multiple user sessions.
2. Congestion Avoidance Testbed Experiment (CATE). This part of the project was done within the framework of the DARTnet experimental T1 national network. The aim of the work was to advance the state of the art in benchmarking DARTnet's performance and traffic control by developing support tools for network experimentation, by designing benchmarks that allow various algorithms to be meaningfully compared, and by investigating new queueing techniques that better satisfy the needs of best-effort and reserved-resource traffic.

This document is the final technical report describing the results obtained by SRI under this project. The report consists of three volumes:

**Volume 1** contains a technical description of the network techniques developed by SRI in the areas of FEC and multicast of real-time traffic (Part 1 of the project).

**Volume 2** describes the work performed under CATE (Part 2).

**Volume 3** contains the source code of all software developed under CATE.

The rest of this volume contains an overview of the work and the results obtained under Part 1. For an overview of the work in CATE, see Volume 2.

## 1.1 ERROR CONTROL FOR HIGH-SPEED NETWORKS

The emerging generation of gigabit wide-area networks (GWANs) will provide long-distance data paths of bandwidth in excess of 1 gigabit/s to individual end users. The fiber-optic links of a GWAN are characterized by very low bit-error rates (BER) on the order of $10^{-14}$. However, in network technologies such as asynchronous transfer mode (ATM), bottlenecks due to limited storage, processing power, and switching speed are likely to cause packet loss. Traditionally, data integrity has been maintained by end-to-end and link-by-link retransmissions. Such *closed-loop* techniques are not effective GWAN paths, because such paths are characterized by a high-bandwidth-delay product. End-to-end retransmissions result in intolerable storage requirements and data delay, and network switches are not expected to have the processing power and storage needed for link-by-link retransmission. For example, assuming data travel at the speed of light, the propagation delay on a 3000 km path is 10 ms, which translates to $10^7$ bit durations at 1 gigabit/s. For these reasons, end-to-end protocols based on *open-loop* control are preferable for GWANs. Open-loop techniques require the use of FEC coding, in which the source adds redundant bits to the transmitted data sequence, to allow a recipient to recover lost or erroneously received data.

A coding scheme for GWAN must have the following properties:

- Recovery from long sequences of lost bits (that is, missing packets)
- Correction of occasional bit errors
- A low probability of undetected errors
- Minimal processing requirements per packet, preferably by low-complexity hardware
- Addition of little delay to the data
- Requirement of minimal amounts of storage.

In Section 2 we present several coding schemes suitable for a GWAN environment. These schemes are based on grouping the data packets into blocks, to which control (parity) packets are added. Each coding scheme is limited in the number of packets it can recover; no packets may be recovered if more than that number are missing from a single block.

Several methods for generating parity packets are described, along with decoding techniques and their hardware-based implementations. We also present algorithms for reducing the effect of *bursts* of missing packets by interleaving the packet sequence. Other techniques for combating burst errors are based on the intelligent management of buffers in the network switches, and the selective discarding of packets from congested buffers to disperse missing packets among as many blocks as possible, thereby reducing the required coding complexity.

Performance evaluation, by both analytic and simulation models, shows that this technique can reduce the packet loss rate by a reduction of up to three orders of magnitude.

## 1.2 HETEROGENEOUS MULTICAST OF REAL-TIME TRAFFIC

Multicast service, in which a source sends information to multiple recipients, has many applications, including the updating of a replicated database, speech and video teleconferencing, electronic mail, newsletter distribution, collaborative environments, and parallel processing.

The need for multicasting was recognized in the early days of computer networking, and some protocols, mainly for multidestination routing, were developed (e.g., reverse-path forwarding of broadcast packets [Karlson and Vetterli 1989]. The emergence of broadband networks and high-performance workstations has provided the opportunity to establish and maintain multicast sessions in which large numbers of real-time streams (e.g., video and multimedia visualizations) are distributed to multiple users.

The user population is expected to be heterogeneous, with the set of multicast recipients greatly differing in their end devices and the network-access bandwidth available to them. Consequently, when a source multicasts a broadband signal, not all intended destinations are willing to receive or are capable of receiving the complete signal. Bandwidth or terminal limitations restrict the rate of information that can be delivered to some, whereas others prefer to pay less and receive only a subset of the information contained in a multicast signal. An example of such a scenario is when a video signal is distributed to a (potentially large) number of recipients who widely differ in their display devices and the bandwidth available to them. Users with wideband access, high-resolution displays, and powerful processors can receive and process the complete high-resolution color video signal, whereas users with less capable displays or lower bandwidth access, who are capable of receiving only part of the signal, may prefer to receive, say, only black-and-white video, rather than receiving no video at all. Similarly, in voice communication, users may settle for low-rate synthetic speech without speaker recognition, when they cannot receive a complete digital speech signal.

Moreover, users may differ in their ability to receive broadband multicast signals, even if they have similar access bandwidth and terminals. In multimedia teleconferencing, users who send and receive multiple streams that represent the various media may not be able to obtain the full bandwidth needed to communicate via all these media simultaneously. Consequently, users must choose the signal to be emphasized at any given time. These decisions, which are made by individual users, are likely to change with time, reflecting the users' ability to focus on different media at different times. For example, users may first allocate most of their access bandwidth for video and de-emphasize computer animation; later, as more computational results are presented through animation, that bandwidth allocation may change to allow a close examination of that signal but reception of only a low-resolution, black-and-white video.

In a multiparty, multimedia session, a number of real-time streams, possibly representing different media, are offered to the participants. In many cases, however, not all participants want to or can receive all of the offered streams at all times. Session flexibility and network efficiency can be greatly enhanced if participants are provided with the means for specifying the streams they wish to receive and the desired quality for each stream. The network can use this information to decide which streams to distribute and where, and to resolve conflicts so as to maximally satisfy the participants' individual requests and capacity constraints. This is particularly important in a heterogeneous network where link capacities and traffic demands vary across the network, thereby necessitating the delivery of different streams to different parts of the network according to individual destinations and network constraints. In such an environment, it is of great importance to optimally distribute multiple streams from a source to a set of destinations, subject to the destinations' requirements and the limitations on available link capacities.

In Sections 3, 4, and 5 we report on three major areas of investigation in heterogeneous multicast (HMC). In Section 3 the basic motivation and techniques for HMC are presented, including an algorithm for computing distribution trees that provide a path from the source to each destination with the maximum available bandwidth. We also present an efficient algorithm for optimal allocation of bandwidth to the various signal layers, based on the bandwidth availability for the destinations.

In Section 4 we focus on the case in which a collection of streams is distributed from a source to the multicast destinations over a precomputed tree. Users' requests are expressed by means of *bids*, which are non-negative values assigned by each destination to each offered stream. Assuming that the source gains an amount equal to a user's bid whenever a stream is delivered to that user, we seek distributions that maximize the sources' gain. The optimal assignment of streams to the links of a multicast tree is formulated as an integer programming problem, and an algorithm for determining the optimal assignment is presented. The algorithm consists of three procedures: listing candidate assignments on a single link, listing candidate assignments on tree paths, and evaluating the gain for each assignment. The algorithm's correctness is proved, and its computational complexity is ascertained.

In Section 5 we explore the issue of maintaining the multicast tree by incrementally changing a maximum bandwidth tree (MBT) for HMC, to accommodate changes in the population of destinations and their traffic demands. The objective is to deliver maximum traffic to each destination while retaining the tree structure and to efficiently distribute traffic. Both centralized as well as distributed asynchronous algorithms are presented. The former algorithms are Dijkstra-like; while the latter are Belman-Ford-like, in that both the cost function and the labelling rule change to suit our objective. Two types of algorithms are presented: in the first, the maximum-bandwidth path to the new destination is obtained on the condition that no changes are made to the existing MBT; in the second algorithm, the maximum-bandwidth path to the new destination is obtained with minimal changes to the existing MBT. For each of the above types of algorithms, we present a centralized asynchronous algorithm of each type, i.e., algorithms whose computations start from the source and destination nodes, respectively. A third algorithm is also presented that restores the tree structure to the routing paths, if that tree structure is destroyed.

# 2 PACKET RECOVERY IN HIGH-SPEED NETWORKS USING CODING AND BUFFER MANAGEMENT

Rapid progress in the development of fiber optics and components for photonic transmission, reception, coupling, filtering, and related communications functions has created the technology for constructing gigabit/s multiuser networks [Nussbaum 1988]. So far most of the research in high-speed communication systems has focused on local and metropolitan area networks (LAN and MAN) [IEEE 1989; Prucnal, Blumenthal, and Santoro 1987; Sun and Gerla 1989; Rom and Shacham 1988; Maxemchuk 1986], and on high-speed switches [Huang and Knauer 1984; Turner 1986; Acampora 1989]. Efforts are now underway to design and construct wide-area networks to span large geographical regions and provide long-distance data paths of bandwidth in excess of 1 gigabit/s to individual end users. We call such systems GWANs.

GWAN data paths are composed of fiber-optic links with very low BERs. It is not uncommon to achieve error rates of $10^{-14}$ over a long-distance fiber cable, despite the presence of regenerators every 100 km.

Despite their enormous link bandwidths, GWANs are still expected to experience bottlenecks due to limited storage, processing power, and switching speed. Statistical resource sharing by a large number of high-speed, fluctuating-rate streams is bound to cause congestion and occasional buffer overflow. Even when a GWAN guarantees users a minimum data rate, its flow-enforcement mechanisms, such as Leaky Bucket [Turner 1986], are designed to delete packets that exceed the network's ability to switch and forward. Packet loss is likely to be the dominant cause of data distortion in GWANs.

Network switches and end users traditionally rely on acknowledgement (ACK)-based closed-loop control to recover lost packets, and adjust incoming traffic rates to overcome congestion [Burton and Sullivan 1972]. The high delay-bandwidth product of long-distance data paths in GWAN results in a large "path storage." For example, assuming data travel at the speed of light, the propagation delay on a 3000 km path is 10 ms, which translates to $10^7$ bit durations at 1 gigabit/s. In such an environment, ACK-based end-to-end protocols require large data storage, reduce channel utilization, and may cause instability. Because of these reasons, end-to-end protocols based on *open-loop* control are being developed for GWANs. An example of this trend is the aforementioned Leaky Bucket flow-control mechanism.

Another example of open-loop control is FEC coding, in which a recipient can recover lost or erroneously received data via parity bits, which are added by the source to the information sequence. Recovering lost packets reduces the need for retransmissions of reliable data, and enhances the quality of real-time data transmission that cannot rely on ACKs and retransmissions because of the large delays involved.

A coding scheme for GWANs must:

- Deal with long sequences of lost bits (that is, missing packets)
- Correct occasional bit errors
- Result in low probability of undetected errors.

Since bits are transmitted at extremely high rates, coding should be done with minimal processing per packet, preferably by low-complexity hardware. It should add little delay to the data and require minimal amounts of storage. In Subsection 2.1 we present several coding schemes suitable for a GWAN environment. These schemes are based on grouping the data packets into blocks, to which control packets are added. Each coding scheme is limited in the number of packets it can recover; no packets may be recovered if more than that number are missing from a single block. In Subsection 2.2 we discuss techniques for reducing the effect of bursts of missing packets by interleaving and by intelligent buffer management, whereby packets are selected for deletion on the basis of their block affiliation. In Subsection 2.3 we use the reduction in packet loss rate as the performance measure for our coding schemes, and discuss the effect of block size, packet arrival rate, and amount of coding overhead on performance. We provide results from both a simple analytic model and more realistic simulations that show the limitations of the coding schemes and the conditions under which they substantially reduce in loss rate.

## 2.1 CODING FOR PACKET RECOVERY

### 2.1.1 General Considerations

When noise-induced bit errors are the main cause of data distortion, as is the case in current computer communication networks, data are carried in the same packet as the error-control bits protecting them. However, recovery of lost packets requires that the error-control bits and the data they protect be carried in separate packets. Chiou and Li [1988] proposed the duplication of each data packet to enhance data survivability in high-loss military data networks. In GWAN, packet loss rate is expected to be sufficiently small to make this 50% rate repetition code quite inefficient.

Our approach utilizes sequence numbers, which are already required by many protocols. Since the order in which packets are sent is denoted by sequence numbers in ascending order, a data recipient identifies missing packets by gaps in the arriving sequence, either upon the arrival of a packet with an out-of-sequence number, or after a pre-determined waiting time. Thus, a missing packet can be considered as a sequence of bit *erasures*, i.e., unreliable bits whose location is known [Blahut 1988]. Several FEC codes efficiently correct erasures, most notably the Reed-Solomon codes [ibid]. However, the special case we consider, in which erasures come in packet-size sequences, allows us to employ simpler techniques for recovering lost data.

The rate at which packets are lost often depends on the rate at which packets are emitted by the source. In particular, packets are more likely to be rejected because of buffer overflow as their rate of arrival to the buffer increases. Since coding increases the volume of traffic entering the network, the data packet loss rate is likely to increase after the control bits are added. Thus, the lower the code overhead (i.e., the higher the code *rate*), the lower is the increase in the packet loss rate. However, using control bits to recover lost packets should bring the packet loss rate below the loss rate at which no coding is used. The ratio of packet loss rate after decoding to the rate at which no coding is used is denoted the *loss ratio*, and it must be smaller than 1:1 for the code to be useful.

In addition to the above consideration, the coding scheme must be suitable for implementation in gigabit/s networks: that is, it should require little processing and even be amenable to hardware implementation in hardware. Both coding and decoding should add a minimal amount of delay to the packets, and should require only small data storage for operation.

Based on the above considerations, we propose to group data packets into *blocks* of predetermined size, and add to each block a number of *parity* packets to contain the error-control bits. The number of parity packets, and their construction, determines the maximum number of data packets that can be recovered. However, any subset of missing packets can be recovered by using the parity packets and the other data packets of the block. The balance of this section presents several methods for constructing parity packets and using them to recover lost packets. For simplicity of presentation, we assume that all packets are $m$ bits long; however, the schemes can also work with variable-length packets. We begin with the simplest technique for recovering a single packet per block.

### 2.1.2 A Single Packet Recovery

To each block of $K$ data packets the source adds an $m$-bit parity packet, whose $i$-th bit is given by

$$c_{K+1,i} = \left(\sum_{j=1}^{K} c_{j,i}\right) \bmod (2) \qquad (1)$$

where $c_{j,i}$ is the $i$-th bit of the $j$-th packet. We denote such a packet a "vertical" parity packet, because if the block is arranged as a rectangular array with packets as rows, $c_{K+1},\, i$ is the sum modulo 2 of the bits in the $i$-th column.

The parity packet is generated by an encoder consisting of $m$ exclusive-or gates (XOR, or symbolically $\oplus$), each of which has its output connected to one of its input ports, as shown in Figure 1. Packets are given, one at a time, to the encoder, with bits $1, 2, \ldots, m$ applied to the input ports of gates $1, 2, \ldots, m$, respectively. At a clock pulse, the decoder output port $i$ represents the sum modulo 2 of bits in position $i$ of the previous packets in the block and bit $i$ of the current packet. There is no need to store the whole block of packets at the source; thus, immediately following its "contribution" to the encoder, a packet can be sent over the network. Following the application of the $K$-th packet, the gates' $m$ output ports contain the parity packet, which is then transmitted.

The recipient of the packet sequence employs a decoder similar in structure to the encoder. For a given block, each arriving packet is applied in parallel to the $m$ input ports of the decoder. Following the application of any subset of $K$ packets from a block, the $m$ output ports contain the remaining packet. If no data packet was lost or excessively delayed in the network, those first $K$ packets are the data packets of the block, in which case the recipient may ignore the parity packet. If, however, data packets are missing from a block, the recipient identifies the location of the missing packets from the gap in the sequence numbers, and considers these packets to be erasures. If only one of the block's data packets is missing, the $K$ packets applied to the decoder include $K-1$ data packets and the vertical parity packet. In this case, the erasure is replaced by the contents of the parity generator's output after those $K$ packets are applied to the decoder.

Notice that the arriving packets need not be delayed at the receiver. Every packet is applied to the parity encoder/decoder, and immediately thereafter can be forwarded. Furthermore, packets may be applied to the decoder in an arbitrary order, since the XOR operation is commutative. This is particularly important for networks that do not guarantee sequenced delivery.

### 2.1.3 Multiple Packet Recovery

When a block protected by a single vertical parity packet is received with two or more packets missing, none of those packets can be recovered. More control data must be added to recover multiple packets in a block. Since recovering an erased packet amounts to solving an equation with one unknown, there must be at least as many parity packets as there are missing packets. Each of those parity packets must add a linearly independent equation to the set.

To recover two packets in a block, we propose to add a *diagonal* parity packet, the bits of which are the modulo-2 sums of bits along block diagonals, as follows:

$$c_{K+2,i} = \begin{cases} \left(\sum_{j=1}^{i} c_{j,k+1-j}\right) \bmod(2) & 1 \le i \le K+1 \\ \left(\sum_{j=1}^{K+1} c_{j,k+1-j}\right) \bmod(2) & \mathrm{K}+2 \le i \le m \\ \left(\sum_{j=1}^{K+m-i+1} c_{i-m+j,m+1-j}\right) \bmod(2) & m+1 \le i \le m+K \end{cases} \qquad (2)$$

Here we assume $K < m - 1$. Similar equations can be written for $K \ge m - 1$. Notice that the diagonal parity packet has $m + K$ bits, compared to $m$ bits in a data or vertical parity packet. If the network requires all transmission units to be of the same size, such as ATM cells [Gonet, Adam, and Coudreus 1986], a diagonal parity packet may have to occupy more than one unit. For $K < m$, a diagonal packet occupies two transmission units.

The encoder for constructing the diagonal parity packet is similar in structure to that shown in Figure 1, except that its registers have $m + K$ storage elements each and the $i$-th packet of the block $i = 1, \ldots, K + 1$ is shifted to the right and placed with its first bit in the $i - 1$-st element. To create a block with two-packet recovery capability, the source employs a vertical encoder, as described below, and a diagonal encoder. The operation is still sequential, in that as soon as a

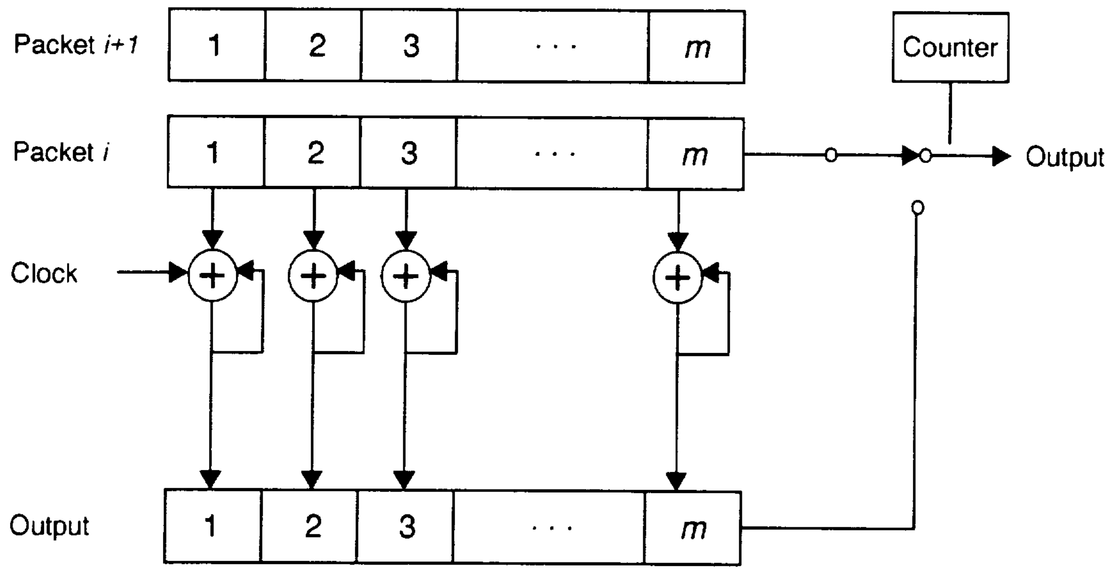


**Figure 1. Parity Packet Generation**

packet is clocked into both encoders (possibly in parallel), it is shipped to the network. Following the clocking of the $K$-th packet, the vertical parity packet is ready, but before it is sent, it is also clocked into the diagonal encoder. At that point, the diagonal packet is ready.

The recipient employs two encoders identical to those used by the source. Its first step in decoding is to generate two new parity packets, one from each encoder, as follows. The data packets of a given block are clocked into each encoder and, as soon as the last one is clocked, the outputs of each encoder are summed modulo 2 bitwise with the corresponding parity packets that arrive from the networks. The new parity packets, one vertical with $m$ and the other diagonal with $m + K$ bits, are used by the recipient to construct up to two missing packets in the block. We denote the bits of these new parity packets $d_{K+1,i}$, $(i = 1, 2, \ldots, m)$ and $d_{K+2,i}$, $(i = 1, 2, \ldots, m + K)$, respectively.

Let us examine how the recipient reconstructs two packets in a block by using the new parity packets. Assume that packets 1 and 2 are missing. The first bit of packet 1, $c_{1,1}$ can be recovered since it equals the first bit of the diagonal packet, $c_{K+2,1}$, and $c_{2,1} = c_{1,1} \oplus d_{K+1,1}$. For $1 < i \leq m$, then $c_{1,i} = c_{2,i-1} \oplus d_{K+2,i}$ and $c_{2,i} = c_{1,i} \oplus d_{K+1,i}$. Decoding of any other pair of data packets is done in a similar fashion. See Figure 2 for a possible hardware implementation of the decoder; note that the interconnections of gates and registers in that figure are for decoding packets 1 and 2, and would be somewhat different for other packet pairs.

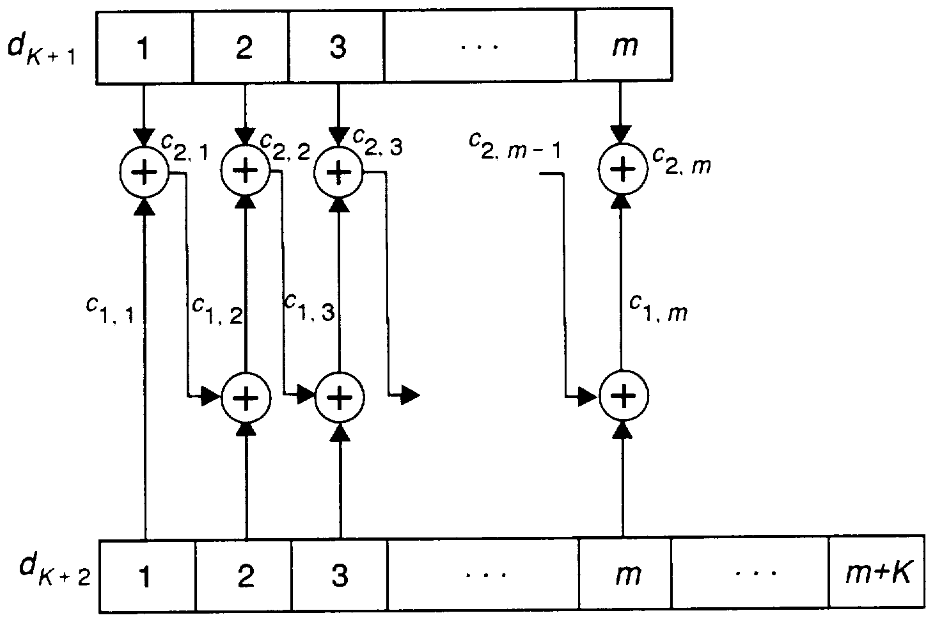


**Figure 2. A Decoder for Reconstructing Two Packets**

If one data packet and one parity packet are missing from an arriving block, the other parity packet is used to recover the data packet. If the only missing packets are the parity packets, the recipient ignores them and forwards only the data packets. If more than two packets are missing from a block, none can be recovered with this 2-packet coding scheme.

It is possible to incrementally enhance the packet-recovering capability by adding parity packets that are constructed by a set of modulo-2 additions, linearly independent of the previously constructed parity packets. For example, to be able to reconstruct three lost packets in a block, we add to the previously described vertical and diagonal parity packets another diagonal parity packet, this one with parity operations along the block diagonals orthogonal to those used in the first diagonal parity packet. The source must now employ an additional encoder, and the decoding scheme is somewhat more complex.

### 2.1.4 Handling Bit Errors

Although bit errors are rare in GWAN, they nevertheless do occur and must be anticipated. A parity bit added to a packet ("horizontal" parity), detects a single bit error in that packet, through a mismatch between its value and the modulo 2 sum of the received data bits. If a single bit error occurs in a block in which no packet is missing, the bit in the vertical parity, which is in the same position in the packet as the erroneous bit, shows a similar mismatch. The intersection of the column and row identifies the bit in error, thereby allowing the recipient to correct it. To detect more than one bit error in a packet, a stronger error detecting code is needed, for example 16-bit CRC [Blahut 1988]. Multiple errors per packet can be corrected by the CRC and the parity packet, provided no packet is missing from that block and no other packets contain errors. This condition is quite reasonable because of the very low error probability in fiber-optic links.

If there are both missing packets and bit errors in a block, and the total number of those packets is not larger than the maximum that can be recovered, the recipient considers the packets with bit errors as erasures and recovers all the packets, as described in the previous subsection.

It is also possible to add a small amount of parity bits that allow the recipient to correct bit errors in a packet and recover the maximum number of erased packets allowed by the parity packets. Consider for example a block of $K$ data packets to which vertical and diagonal parity packets, as described in the previous subsection, are added. Suppose that in such a block packets 1 and 2 are missing and in packet 3 bit $c_{3,i}$ is erroneous.

The recipient first attempts to recover packets 1 and 2 as if packet 3 does not include a bit error. This results in an error pattern consisting of two bit errors in packet 2 and a single error in packet 1. To correct these errors the recipient needs additional $m/2$ parity bits. Each such bit includes in its construction only one of the errors in the above pattern. By knowing the packets in which the error occurred, two diagonal parity packets are sufficient to determine the location of the errors. Notice that these $m/2$ parity bits can be accommodated in the second part of the diagonal parity packet, where $K < m/2$, so that no additional packets are generated.

### 2.1.5 Recovering a Burst of Lost Packets

Arranging the data in a matrix, and adding parity bits both for the rows and the columns, can used for correcting bit errors [Clark and Cain 1981]; this idea also be employed to recover bursts of lost packets. To do so, we arrange the data packets in a $K \times M$ array and add to each row and each column of the array a parity packet. The $i$-th bit of a parity packet is the modulo 2 sum of the $i$-th bits of the data packets in the corresponding row or column. That is, if we denote by $c_{i,j,k}$ the $k$-th bit of the packet in the $(i,j)$ position in the array, then

$$c_{i,M+1,k} = \left(\sum_{j=1}^{M} c_{i,j,k}\right)\text{mod}(2), \qquad 1 \leq k \leq m, \quad 1 \leq i \leq K \tag{3}$$

$$c_{K+1,j,k} = \left(\sum_{i=1}^{K} c_{i,j,k}\right)\text{mod}(2), \qquad 1 \leq k \leq m, \quad 1 \leq j \leq M \tag{4}$$

The code rate, $(K+M)/(KM)$, can be varied by adjusting the dimensions of the array.

If the packets are sent by rows, this scheme can recover bursts of missing packets of any length less than or equal to $M$. Such recovery is done with the aid of the column parity packets only. The row parity packets are used to recover additional missing packets scattered over the array. This technique requires larger hardware complexity: $K + M$ coders similar to those described in Subsection 2.1.3. Note that packets do not encounter larger delays at the source than they would with no interleaving. Also, if a row suffers only a single missing packet, that packet can be recovered after the last packet of the row is received. On the other hand, if two or more packets are missing from a row, they must wait until their respective columns are fully received.

## 2.2 BUFFER MANAGEMENT AND INTERLEAVING

Bit errors are caused by random processes; and although network designers can affect the average error rate, say by adjusting signal power, they cannot select the bits that will be received in error. In contrast, a congested node selects a particular packet for deletion from the set of packets available in its buffer, according to preprogrammed buffer management rules. In current networks, these rules are designed to support congestion control and improve throughput and fairness in service. For example, a buffer management scheme may give priority to packets that are already in the buffer and may reject all packets that arrive when the buffer is full. Other rules may assign priority to packets based on their source or destination, on the elapsed time they have spent in the network, or according to the rate at which those packets are emitted by their source.

Buffer management rules may not affect the average rate at which packets are lost, but they can have a strong effect on the distribution of lost packets. The performance of the error control scheme described above strongly depends on this distribution. For example, two missing packets in a block are not recoverable, but two packets in adjacent blocks are. Thus, the role of buffer management procedures in enhancing end-to-end error control can be that of dispersing the deleted packets so as to minimize the number of blocks that arrive with multiple missing packets. This can be done in the following manner. When an arriving packet finds a full buffer and that packet's block has already suffered a lost packet, the server deletes from its buffer a packet from a previous block that has not lost any packets so far and admits the arriving packet. Only if such an intact block cannot be found does the server delete a second packet from a block. Figure 3 depicts the algorithm by which packets are deleted from the buffer.

Suppose now that the server in question has a finite buffer that can store up to $B$ packets, and that the arrival process consists of a stream of packets sent in ascending sequence numbers from a single source. Making the block size $K$ less than $B$ guarantees that whenever an arriving packet finds a full buffer, there is at least one packet from a previous block in the buffer. If $B \leq K$, arriving packets from the end of the $i$-th block may not find packets from the $i-1$-st block in the buffer, thereby limiting the usefulness of this technique. In the next section we show that although the best performance without buffer management is sometimes achieved for a block length larger than the buffer size, not much degradation is encountered when $K$ is restricted to be smaller than or equal to $B$. This provides for further improvement by buffer management.

In the discussion so far we have assumed that the packet stream arriving at the server consists of a consecutive sequence of contiguous packet blocks. Under these circumstances, the server has access at any moment to at most $\lceil B/K \rceil$ blocks, aside from the block whose packets are arriving at that moment. For a single packet recovery coding, $\lceil B/K \rceil + 1$ is the maximum-length burst of missing packets that can be recovered.

If the level of spreading offered by the buffer management is not sufficient, the source can arrange its packets in $N$ interleaved streams, by assigning to each stream every $N$-th packet. Such *deterministic interleaving* spreads bursts of deleted packets in the arriving stream over multiple blocks. However, monitoring multiplexed streams requires more effort by the server than is needed for a single stream.

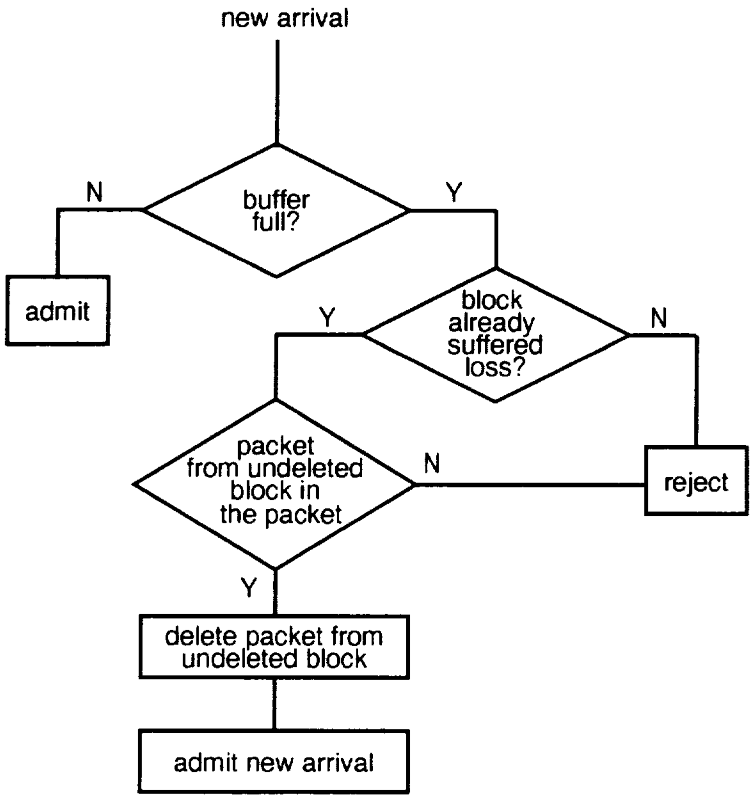

new arrival
buffer full?
N
Y
admit
block already suffered loss?
Y
N
packet from undeleted block in the packet
N
reject
Y
delete packet from undeleted block
admit new arrival

**Figure 3. An Algorithm for Packet Rejection**

"Natural" spreading of bursts occurs when the arriving stream comprises intermixed packets from many sources. This is the case, for example, when the server represents a controller of an output queue in a space-division switch [Hluchyj and Karol 1988], to which packets are arriving from all input ports, each carrying an independent stream. In this case the packet mix is random rather than deterministic, in the sense that a given packet belongs to a specific stream with some probability. It is interesting to note that deterministic interleaving offers better burst spreading and thus lower probability of loss than random interleaving. For example, consider a burst of length $s \leq N$. All the packets in this burst are recoverable under deterministic interleaving because no block suffers more than one missing packet. On the other hand, in the random case, there is some probability that some blocks suffer two or more losses. This probability may be quite large, a phenomenon known as the "birthday paradox" [Feller 1958], so named because of the "paradoxical" fact that among a group of 23 people, there is a better than 50% probability that two of them will share a birthday. In our case, a burst of 14 consecutive erasures in a data stream consisting of 100 randomly multiplexed streams results in a 0.615 probability that two of the erasures will share the same traffic stream, thereby resulting in an unrecoverable erasure. In contrast, the use of a 100-way deterministically interleaved stream would guarantee that each of the erasures in the burst of 14 would occur in a different FEC block.

## 2.3 PERFORMANCE EVALUATION

As indicated above, the two factors in the packet loss process that must be incorporated in a model are

- The packet rejection distribution and the effect of adding parity packets on that distribution
- The reduction in packet loss rate through buffer management and coding.

The model we use in this section for ascertaining this effect is depicted in Figure 4. It consists of a data source to generate both data and parity packets and send them through a single server with a finite buffer, which represents the network. Packets are lost when an arriving packet finds a full buffer. The specific packet that is rejected in this case depends on the buffer management scheme exercised by the server. The data recipient gets the packet sequence and attempts to recover missing data packets by using the parity packets.

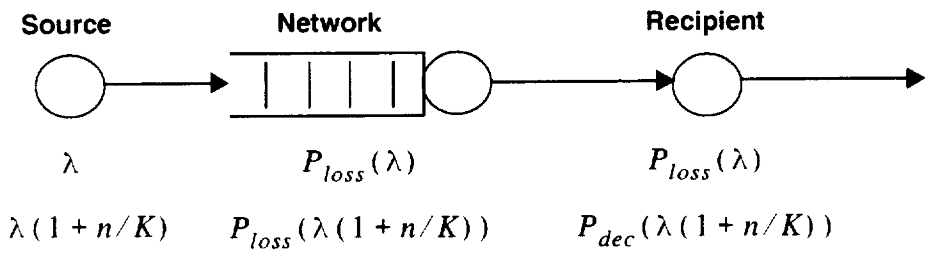


**Figure 4. Performance Evaluation Model**

The main parameters in the model are the buffer size ($B$), the number of parity packets per block ($n$), the number of data packets in a block ($K$), and the rate, denoted by $\lambda$ packets/second, at which the source generates fixed-size ($m$ bits) data packets. Time is slotted and each data packet requires exactly one slot to be transmitted. We assume that the number of packets arriving during slots 1, 2, ..., are independent Poisson distributed random variables with mean $\lambda$. The measure of performance we use is the loss ratio, defined in the following paragraph.

When only data packets are sent, at rate $\lambda$, the loss rate $P_{loss}(\lambda)$ experienced at the server is also the loss rate observed at the recipient output assuming no retransmissions. Adding $n$ parity packets to every block of $K$ data packets increases the packet rate at the server to $\lambda(1 + n/K)$ and consequently the loss rate of the packet stream to $P_{loss}(\lambda(1 + n/K))$. This latter loss rate, however, is reduced by decoding at the recipient to $P_{dec}(\lambda(1 + n/K))$. The loss ratio, $G$, of the coding scheme is defined as

$$G \equiv \frac{P_{dec}(\lambda(1 + n/K))}{P_{loss}(\lambda)} \tag{5}$$

and for a coding scheme to be useful, it must have $G < 1$.

Several interesting tradeoffs can be investigated with this model. An example is the percent of parity bits in the packet stream. On the one hand, the number of packets that can be recovered in a block increases with the number of parity packets in the block. On the other hand, a large percent of overhead packets increases the traffic rate and hence the loss rate in the packet stream arriving at the decoder. The percent of overhead packets is also constrained by the requirement that the total packet rate, data and parity, should be less than 1 to avoid buffer saturation. We investigate these and other considerations in the subsections below with a simple analytic model and with more realistic simulation results. We end this section with a discussion of FEC code design constraints.

### 2.3.1 Analytic Model

To analytically model the performance, we consider the case in which the packets arriving at the server are from a single source, and coding is based on adding a single parity packet to each block of $K$ data packets with no interleaving. We assume that the numbers of packets arriving in time slots 1, 2, ... are independent, Poisson distributed random variables with rates $\lambda$ and $\lambda(1 + n/K)$, for the uncoded and coded packet streams respectively.

The packet loss probabilities, $P_{loss}(\lambda)$ and $P_{loss}(\lambda(1 + n/K))$ are each the rejection probability for a discrete-time, single-server queue with finite size $B$, and constant service time of one slot. The buffer behavior is modeled as a finite-state discrete time Markov chain, in which the state is the number of packets in the queue just before the beginning of a slot. The state transition probabilities can be found in the literature (for example by Hluchyj and Karol [1988]).

The decoder performance is evaluated under an additional assumption, which is that each packet finds a full buffer with probability $p = P_{loss}(\lambda(1 + n/K))$, independently of other packets. That is, we represent the effect of buffer overflow by marking each packet with a "loss" tag with probability $p$ and leave it unmarked with probability $1 - p$.

First consider decoding with no buffer management. At the decoder, the number of lost packets in a block is a random variable with binomial distribution and parameters $K+1, p$. That is,

$$Pr\{ i \text{ lost packets in a block} \} = b(K+1, i, p) \equiv \binom{K+1}{i} p^i (1-p)^{K+1-i} \tag{6}$$

A lost packet can be recovered if and only if it is the only lost packet in its block. The average number of packets lost in a block after decoding is given by

$$EL = \sum_{j=2}^{K+1} jb(K+1, j, p) = (K+1)p - (K+1)p(1-p)^K \tag{7}$$

The packet loss rate after decoding, $P_{dec}$, is thus given by

$$P_{dec} = EL/(K+1) = p - p(1-p)^K \approx p(1 - e^{-Kpp}) \tag{8}$$

where the approximation is for small values of $p$ and large values of $K$ such that $Kp$ is finite.

Let us now consider the selective rejection of packets at a typical network node. Suppose that the node buffer is not smaller than the block length, so that whenever an arriving packet, say from block $l$, finds a full buffer, there is at least one packet from block $l-1$. The new packet is accepted into the buffer only if a previous packet from block $l$ has been rejected and block $l-1$ did not suffer any packet loss. In this case, a packet from block $l-1$ is rejected to make room for the new arrival from block $l$. This implies that block $l-1$ arrives with no lost packets if and only if block $l$ has no more than one lost packet.

To ascertain the effect of this buffer management procedure on the decoder's performance, we start with the stream of rejected ("marked") and accepted packets. If two or more packets are missing from block $l$ and none from $l-1$, the buffer management action amounts to deleting a packet from block $l-1$ and restoring a packet in block $l$. If a packet is already missing from block $l-1$, no "trading" is done.

Let $i, j$ be the quantity of missing packets in blocks $l$ and $l-1$, respectively. Let $L(i, j)$ be the number of packets that cannot be recovered after decoding in block $l$, given $i, j$.

$$L_{i,j} = \begin{cases} 0 & i = 0, 1 \\ 0 & i = 2, j = 0 \\ i-1 & i \geq 3, j = 0 \\ i & i \geq 2, j > 0 \end{cases} \tag{9}$$

Thus, the probability of packet loss after decoding with buffer management is given by

$$P_{dec} = EL/(K+1) =$$

$$\frac{1}{(K+1)} \sum_{i=3}^{K+1} \{ib(K+1,i,p)\,[1-b(K+1,0,p)] + (i-1)\,b(K+1,i,p)\,b(K+1,0,p)\}$$

$$= p - \frac{b(K+1,1,p)}{K+1} + \frac{b(K+1,0,p)}{K+1} \cdot [1-b(K+1,1,p) - b(K+1,0,p) + b(K+1,2,p)] \tag{10}$$

Tables 1 and 2 depict the performance of the scheme described above, for buffer size $B$=20, and packet arrival rates before encoding of 0.8 and 0.9, respectively. The entries for $K = \infty$ represent no encoding, and the packet loss probabilities are those for a single server queue with the above arrival rates. The second row shows the packet loss rate for the value of $K$ that minimizes that loss rate after decoding, without buffer management. For example, for $\lambda = 0.8$, with no encoding the packet loss rate is $2.55 \times 10^{-5}$. With block size $K = 70$, the higher packet rate of $0.8\,(1 + 1/70)$ causes the loss probability to increase to $4.17 \times 10^{-5}$. However, the decoder reduces that rate to $1.22 \times 10^{-7}$, thereby achieving a total packet loss reduction of more than two orders of magnitude. For $K = 19$, the largest block size for which the server can apply the buffer management scheme effectively, the total improvement in $P_{dec}$ is almost three orders of magnitude. At data-packet rate $\lambda = 0.9$, the initial loss rate is higher than before and the improvement is smaller, as Table 2 shows. However, even at that congested level, the packet recovery scheme, along with proper buffer management, reduces the packet loss rate by more than an order of magnitude.

**Table 1. Probability of Packet Loss with $B = 20$ and $\lambda = 0.8$; All Quantities Are for $\lambda(1 + 1/K)$**

| $K$ | LOSS PROBABILITY | | |
|---|---|---|---|
| | BEFORE DECODING | AFTER DECODING | WITH BUFFER MANAGEMENT |
| ∞ | $2.55 \times 10^{-5}$ | | |
| 70 | $4.17 \times 10^{-5}$ | $1.22 \times 10^{-7}$ | |
| 19 | $1.5 \times 10^{-4}$ | $4.25 \times 10^{-7}$ | $1.28 \times 10^{-4}$ |
| 10 | $6.55 \times 10^{-4}$ | $4.28 \times 10^{-6}$ | $2.51 \times 10^{-7}$ |
| 8 | $1.35 \times 10^{-3}$ | $1.46 \times 10^{-5}$ | $1.12 \times 10^{-6}$ |
| 6 | $4.11 \times 10^{-3}$ | $1.0 \times 10^{-4}$ | $1.15 \times 10^{-5}$ |

Table 2. Probability of Packet Loss with $B = 20$ and $\lambda = 0.9$

| $K$ | LOSS PROBABILITY | | |
|---|---|---|---|
| | BEFORE DECODING | AFTER DECODING | WITH BUFFER MANAGEMENT |
| ∞ | $1.35 \times 10^{-3}$ | | |
| 65 | $2.18 \times 10^{-3}$ | $2.89 \times 10^{-4}$ | |
| 19 | $6.28 \times 10^{-3}$ | $7.07 \times 10^{-4}$ | $1.22 \times 10^{-4}$ |
| 10 | $1.89 \times 10^{-2}$ | $3.26 \times 10^{-3}$ | $8.78 \times 10^{-4}$ |
| 8 | $3.0 \times 10^{-2}$ | $6.39 \times 10^{-3}$ | $2.11 \times 10^{-3}$ |
| 6 | $5.43 \times 10^{-2}$ | $1.51 \times 10^{-2}$ | $6.39 \times 10^{-3}$ |

### 2.3.2 Simulation

In the analytic model we assume that packets are rejected independently. To study the sensitivity of the coding scheme to correlation in the rejection process, we simulated the model shown in Figure 4. Runtime parameters to the simulation include data input rate, the type of FEC, the amount of deterministic interleaving or random multiplexing, and the type of buffer management.

Each simulation run was repeated five times with different random number generator seeds. In all cases, the confidence intervals computed from these values are more narrow than the lines on the graphs. Although the effects of initial conditions were not eliminated, comparisons with analytic results, where available, indicate that the measured simulation results deviate by less than 10% from the computed values.

The following paragraphs compare the results of the simulation with those of the analytic model, then present simulation results comparing the efficacy of buffer management, FEC, deterministic interleaving, and random multiplexing. These results show that buffer management is necessary to achieve an acceptable loss ratio, that a simple single-erasure-correcting FEC scheme used in conjunction with deterministic interleaving is superior to more complex FEC schemes, that deterministic interleaving is superior to random multiplexing, and that loss ratios of $10^{-3}$ are achievable.

### 2.3.2.1 Comparison to Analytic Results

Tables 3 and 4 show that the loss probability, both with and without buffer management, is much worse than that predicted by the analytic model; in fact, in all cases the FEC gain is insufficient to overcome the greater erasure rate caused by the addition of the redundant packets. This is due to the extremely bursty nature of the queue overflow process. The following sections evaluate some methods for improving this situation.

**Table 3. Probability of Packet Loss with $B$ = 20 and $\lambda$ = 0.8 (Simulation)**

| $K$ | LOSS PROBABILITY | | |
|---|---|---|---|
| | BEFORE DECODING | AFTER DECODING | WITH BUFFER MANAGEMENT |
| $\infty$ | $2.68 \times 10^{-5}$ | | |
| 70 | $4.79 \times 10^{-5}$ | $3.96 \times 10^{-5}$ | $3.45 \times 10^{-5}$ |
| 19 | $1.8 \times 10^{-4}$ | $1.53 \times 10^{-7}$ | $8.53 \times 10^{-5}$ |
| 10 | $6.67 \times 10^{-4}$ | $5.15 \times 10^{-4}$ | $1.98 \times 10^{-4}$ |
| 8 | $1.61 \times 10^{-3}$ | $1.21 \times 10^{-3}$ | $3.36 \times 10^{-4}$ |
| 6 | $3.85 \times 10^{-3}$ | $2.77 \times 10^{-3}$ | $5.61 \times 10^{-4}$ |

**Table 4. Probability of Packet Loss with $B$ = 20 and $\lambda$ = 0.9 (Simulation)**

| $K$ | LOSS PROBABILITY | | |
|---|---|---|---|
| | BEFORE DECODING | AFTER DECODING | WITH BUFFER MANAGEMENT |
| $\infty$ | $1.61 \times 10^{-3}$ | | |
| 65 | $1.87 \times 10^{-3}$ | $1.74 \times 10^{-3}$ | $1.66 \times 10^{-3}$ |
| 19 | $5.78 \times 10^{-3}$ | $5.14 \times 10^{-3}$ | $3.89 \times 10^{-3}$ |
| 10 | $1.47 \times 10^{-2}$ | $1.23 \times 10^{-2}$ | $6.69 \times 10^{-3}$ |
| 8 | $2.41 \times 10^{-2}$ | $1.95 \times 10^{-2}$ | $9.34 \times 10^{-3}$ |
| 6 | $4.68 \times 10^{-2}$ | $3.69 \times 10^{-2}$ | $1.45 \times 10^{-2}$ |

### 2.3.2.2 Buffer Management

The great improvement due to buffer management is clearly illustrated in Figure 5, which shows the loss ratio with and without buffer management, respectively, for a M/D/1/10 queue carrying randomly multiplexed streams with an aggregate $\lambda$ of 0.85. Each stream has single-erasure-correcting FEC, each block of which consists of 20 data packets and a single FEC packet. A loss ratio of zero indicates perfect performance.

As can be seen from the figure, buffer management improves the loss ratio by up to two orders of magnitude. Note that more than two traffic streams must be present in order for coding to show any benefit at all, even with buffer management. As will be shown later, deterministic interleaving may be used to overcome this difficulty.

### 2.3.2.3 FEC Versus Deterministic Interleaving

Another method of reducing the loss ratio is to increase the erasure-correcting capability of the FEC coding scheme, by using the diagonal parity packets described in previous sections. Recall that a diagonal parity packet is longer than a data packet. We make the pessimistic assumption that two MB-bit packets are added to the traffic for each diagonal packet.

This section compares three FEC coding schemes: a single-erasure-correcting scheme requiring one overhead packet per block, a double-erasure-correcting scheme requiring three overhead packets per block, and a triple-erasure-correcting scheme requiring five overhead packets per block. These schemes are compared at equal code rates; thus, the first scheme has 20 data packets per block, the second has 60, and the third has 100.

The single-erasure-correcting scheme with a three-way interleave can be considered to act as a separate coding scheme, with 60 data packets per block, that is capable of correcting from one to three erasures per block, depending on the distribution of erasures within the block. Similarly, the single-erasure-correcting scheme with a five-way interleave can be considered to act as a separate coding scheme, with 100 data packets per block, that is capable of correcting from one to five erasures per block, again depending on the distribution of erasures within the block.

Figure 6 demonstrates that interleaving the single-erasure-correcting scheme is superior to using the more complex multiple-erasure-correcting schemes for a M/D/1/10 queue with buffer management and $\lambda = 0.85$. The loss ratio for each FEC coding scheme is plotted against a normalized degree of interleaving, consisting of the actual degree of interleaving multiplied by the ratio of the block length divided by the block length of the single-erasure-correcting scheme.

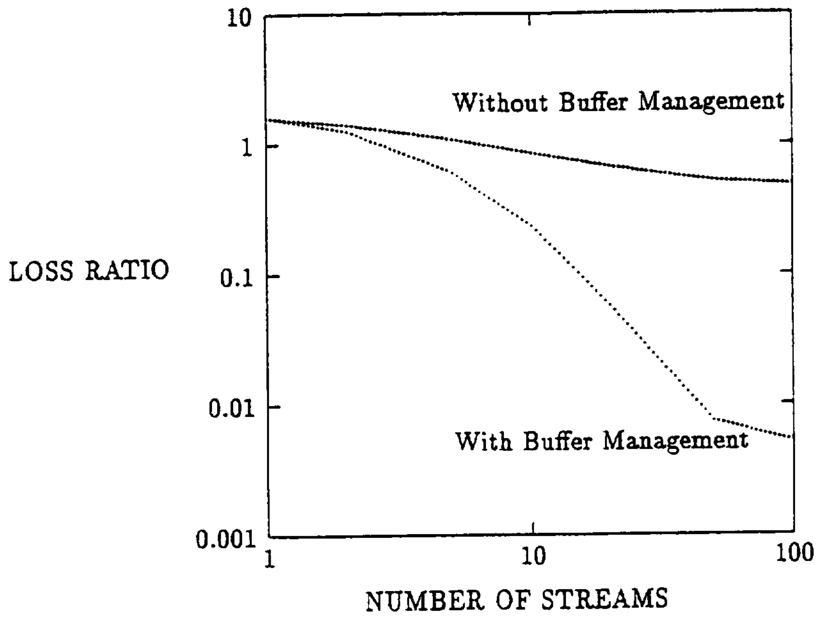


**Figure 5. Loss Ratio with and without Buffer Management**

Figure 6 thus compares the double-erasure-correcting scheme to the single-erasure-correcting scheme at triple the interleaving, and compares the triple-erasure-correcting scheme to the single-erasure-correcting scheme at quintuple the interleaving.

Note the presence of loss ratios of better than $10^{-3}$ in Figure 6.

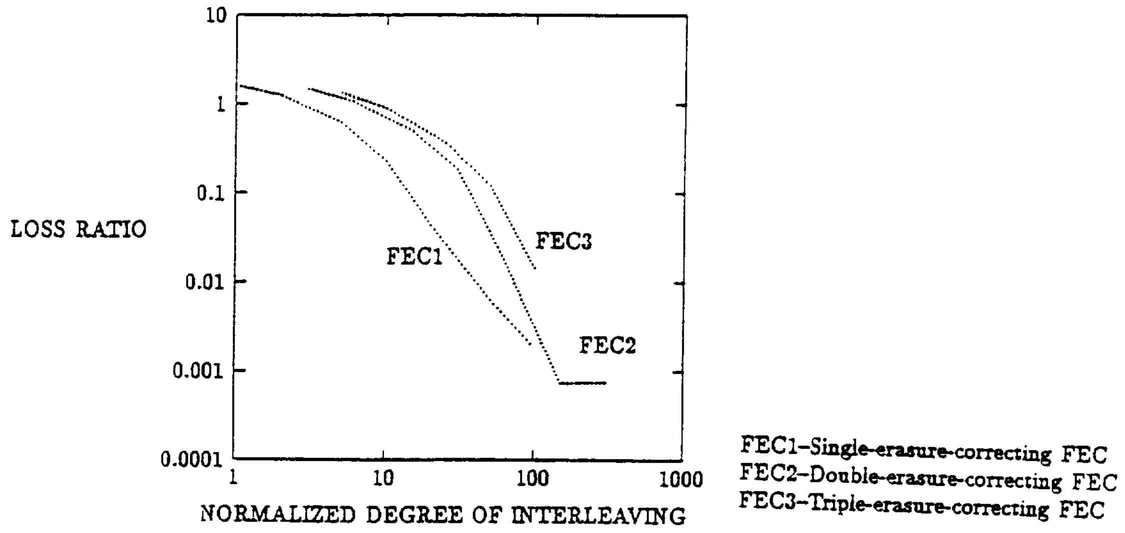


**Figure 6. FEC Versus Deterministic Interleaving**

### 2.3.2.4 Interleaving Versus Multiplexing

Figure 7 shows that deterministic interleaving is slightly superior to random multiplexing for an M/D/1/10 queue with buffer management and load $\lambda = 0.85$. This effect becomes more pronounced for larger queues, which have lower raw loss rates.

However, the increased interleaving results in increased delay for the correction of erased packets. The tradeoff between delay and reliability is application dependent. It is instructive to compare the delay caused by interleaving with the delay imposed by round-trip time on

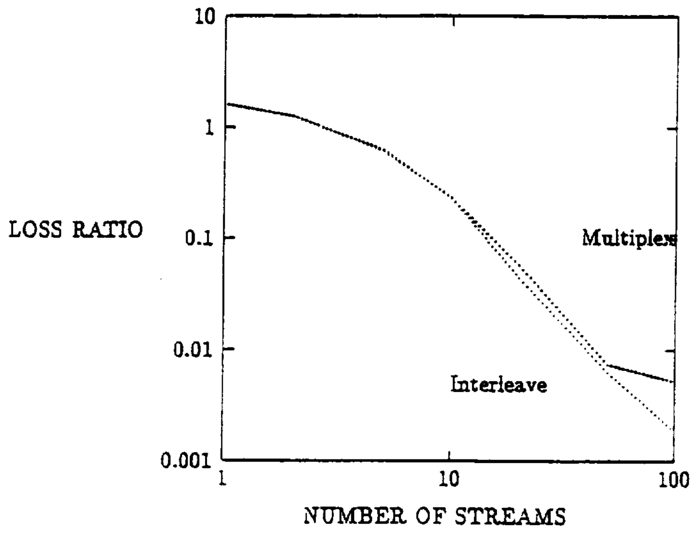


**Figure 7. Interleaving Versus Multiplexing**

retransmissions. Assuming that the source and destination are located 3000 km from each other, communicate at a rate of 1 gigabit/s, and are connected by a speed-of-light communications medium, the round-trip time will be greater than 1600 packet durations. This means that an FEC scheme that uses a block length of 20 packets may use up to 160-way interleaving, while still maintaining an average reconstitution delay smaller than is possible with a retransmission-based scheme.

## 2.4 CONCLUSION

The novel technique for reducing the packet loss rate in high-speed wide-area networks, which was presented in this section, is based on open loop recovery. Parity packets, which the source adds to each block of data packets, are used by the recipient to reconstruct lost packets from the respective blocks. The missing packets are identified by observing a sequence-number gap in the stream of incoming packets. Methods for recovering a single or double loss, and larger numbers of losses from a block were presented. Bit errors can be corrected by either reconstructing the affected packet as if it were missing or by additional parity information.

To alleviate the effect of packet loss correlation, we proposed to enhance buffer management procedures in the networks so that packets are rejected based on their block affiliation and the number of packets already lost in their block. A similar effect can be achieved by interleaving the data, either intentionally by the source, or as it occurs naturally during statistical multiplexing.

We evaluated the performance of these packet recovery schemes by using a model, which consists of a source that codes the data; a single-server, discrete-time, finite-capacity queue which causes packet loss and at which buffer management is carried out; and a recipient, which uses a decoder to reconstruct missing packets. The performance measure we selected was the ratio of packet loss rate after decoding to the packet loss rate when no coding is used. This ratio, denoted loss ratio, must be smaller than 1 for the code to be useful. Both analytic and simulation results were presented for this model. The former, obtained under the assumption of independent packet losses, showed loss ratios of the order $10^{-3}$ and smaller.

The simulation runs showed that packet loss correlation is a severe problem when each packet stream has a dedicated finite buffer. In this case, buffer management and/or interleaving is vital to the achievement of good loss ratios. We have also shown that deterministic interleaving achieves better loss ratios than statistical multiplexing. This observation implies that when multiple streams are intermixed, as in output queues of a space-division switch, round-robin polling is superior to random polling. Loss ratios of better than $10^{-3}$ were demonstrated.

In summary, our performance study shows that a significant reduction in packet loss rate can be achieved with a combination of coding, buffer management, and interleaving.

# 3 MULTIPOINT COMMUNICATION BY HIERARCHICALLY ENCODED DATA

Multicast service, in which a source sends information to multiple recipients, has many applications, including the updating of a replicated database, speech and video teleconferencing, electronic mail, newsletter distribution, collaborative environment, and parallel processing.

The need for multicasting was recognized in the early days of computer networking, and some protocols, mainly for multidestination routing, were developed (e.g., reverse-path forwarding of broadcast packets [Karlsson and Vetterli 1989]. Recently, new techniques have also emerged, including multicast routing [Ghanbari 1989; LeGall 1991], reliable end-to-end packet delivery [Burton and Sullivan 1972; Schwartz 1987; Shacham and McKenney 1990; Acampora 1989], and multicast switching [Bailly et al. 1980; Blahut 1988].

Early generations of data communication networks offered only limited multicast service, since their narrow-band links, small user populations, and limited computation and display capabilities made such services infeasible or unattractive for many applications. However, the emergence of broadband network technology, such as the asynchronous transfer mode, opens new opportunities for multicast services. Broadband channels and packet-based fast switching allow many more users than in the past to connect to networks, facilitate more types of interaction among users, and support a larger variety of traffic types including broadband signals such as video. This combination of physical means for supporting multiple services and a large interconnected user base that can benefit from such services provides a fertile ground upon which a rapid growth of multicast services can be expected [Lee, Boorstyh, and Arthurs 1988; Bailly et al. 1980].

The user population is expected to be heterogeneous, with the set of multicast recipients greatly differing in their end devices and the network-access bandwidth available to them. Consequently, when a source multicasts a broadband signal, not all intended destinations are willing to receive or are capable of receiving the complete signal. Bandwidth or terminal limitations restrict the rate of information that can be delivered to some, whereas others prefer to pay less and receive only a subset of the information contained in a multicast signal. An example of such a scenario is when a video signal is distributed to a (potentially large) number of recipients with widely different display devices and available bandwidths. Users with wideband access, high-resolution displays, and powerful processors can receive and process a complete high-resolution color video signal, whereas users with less capable displays or lower bandwidth access, who can receive only part of the signal, may prefer to receive, say, only black-and-white video to receiving no video at all. Similarly, in voice communication, users may settle for low-rate synthetic speech without speaker recognition when they cannot receive a complete digital speech signal.

Moreover, users may differ in their ability to receive broadband multicast signals, even if they have similar access bandwidth and terminals. In multimedia teleconferencing, users who send and receive multiple streams that represent the various media may not be able to obtain the full bandwidth needed to communicate via all these media simultaneously. Consequently, users must choose the signal they emphasize at any given time. These decisions by individual users are likely to change with the users' ability to focus on different media at different times. For example, users may first allocate most of their access bandwidth for video and de-emphasize computer animation;

later, as more computational results are presented through animation, that bandwidth allocation may change to allow a close examination of that signal, but reception of only a low-resolution, black-and-white video.

All existing multicast protocols were developed on the assumption that all recipients receive all the information emitted by a source. Using such protocols in a heterogeneous environment raises a serious dilemma: a session must either exclude the more limited users who cannot receive the full signal, or must penalize the more capable users by compressing (and distorting) the signal to fit the least capable users.

In this section we present an approach for multicasting in a heterogeneous environment, where each destination receives the subset of the signal allowed by its constraints. This approach requires careful signal selection and coordination among the source, destination, and network. In this paper we focus on the following areas of the proposed approach.

**Signal Representation.** The source's signal is encoded and presented to the network as a set of bit streams called layers. The layers are so organized that the quality of reception is proportional to the number of layers received—the first layer provides the basic information, and every layer improves upon the quality of the layers below it. (Such coding algorithms are known as subband or hierarchical coding, and the corresponding signal representations are known as layered or hierarchical signals.) We then review several layered coding techniques and analyze their suitability for multicast service.

**Routing and Signal Optimization.** The network employs algorithms that find the maximum bandwidth available to each destination and compute optimal paths with those bandwidth values. We present

- An efficient procedure for determining the maximum bandwidth to each destination, given that all the layers to each destination must be sent on the same path
- A signal optimization procedure that, given the set of available bandwidths, assigns bandwidth to each layer so as to maximize the signal quality as seen by the set of destinations
- An algorithm for determining the maximum-bandwidth, shortest paths to each destination.

**Maintaining Data Integrity.** The source's signal is protected by packets carrying error-control bits, which are sent along with the data packets and allow a receiver to recover lost data packets. The layers of the signal are individually protected, since missing packets at different layers have different effects on the received-signal quality, and since destinations that receive only a subset of the signal cannot make use of error-control data that are applied to the whole signal as a single stream. Moreover, it is expected that many applications of the type of multicast discussed here will require time-constrained delivery of data, thereby excluding retransmission as a data recovery mechanism. We also present techniques for open-loop recovery of lost data based on erasure correction, which was presented in Section 2, and discuss ways of providing different protections to different layers of a signal.

## 3.1 HIERARCHICAL ENCODING OF DATA

*Hierarchical encoding* is a name for a recently developed family of signal representation techniques, in which the source information, most commonly a digital real-time signal, is partitioned into substreams, each of which represents a well-defined portion of the signal.* The substreams, also known as *layers*, are so constructed that substream 1 (the lowest layer) carries the elements that are essential for reconstruction of the signal by the receiver, although the resulting signal may be of low quality. Substream $i$ ($i > 1$) contains information that improves the quality of reception.

Extensive research has been conducted on hierarchical encoding, especially for video and voice signals. The first hierarchical encoding scheme was designed for speech transport over packet-switching integrated networks [Burton and Sullivan 1972]. In this technique, the lowest layer contains the most significant bits of the digital representation of the speech signal, and Layer $i$ contains bits of lesser significance than Layer $i$-1, but of greater significance than those in Layer $i$+1. In this case Layer $i$ improves the signal quality at the receiver, if and only if all layers below it are received as well.

While this technique is also applicable for video, several other hierarchical coding techniques have been developed that exploit the unique features of the video signal, which is composed of a sequence of frames with intraframe spatial correlation and interframe temporal correlation. We describe below two of the numerous hierarchical coding schemes that can be found in the literature, with emphasis on the features that impact signal transmission over a network.

A basic video encoding technique called *conditional replenishment* is utilized to generate a variable bit rate (VBR) stream, based on which a receiver reconstructs a video signal of constant quality [Clark and Cain 1981]. This video coding scheme, however, is sensitive to bit errors and data loss, which suggests that quality may be severely affected when such a stream is transmitted over a packet-switching network. To solve this problem, the video stream is partitioned into two substreams:

1. The first part contains essential video information such as synchronization pulses and address changes, as well as basic video data. Reception this part alone enables the receiver to obtain a video signal. Since this information is vital, it must be received with high reliability.
2. The second part contains "add-on" information, which improves the quality of the received video. This information can be sent over the network over shared links, which results in some packet loss. However, the video signal is separated in such a way that losses in the second part do not affect the quality of the first part.

This scheme is implemented with 110-120 Kbps for the complete video signal, of which 24 Kbps are devoted to the first part [ibid]. When both parts of the signal suffer no losses, the picture quality is dependent only on the coding parameters of the second part. As the loss rate of the second part increases, the video exhibits graceful degradation in quality. Even at 100% packet loss rate for the second layer, the signal exhibits reasonable quality despite being somewhat impaired by smearing and block structure distortion.

*Hierarchical encoding is also known as pyramidal, layered, or subband encoding.

*Hierarchical encoding* is based on partitioning the spectrum of the video signal along its three-dimensional frequency region: horizontally, vertically, and temporally [Chiou and Li 1988]. Region 1 contains the low-pass components of the signal and is the only subband that is essential for signal reception. Receiving only subband 1 results in a video signal of low quality; every other layer adds to the quality of the signal. The simulation of subband coding using 11 frequency regions showed that the various subbands greatly differ in their mean rate and burstiness [ibid]. Subband 1, for example, was generated at 600 Kbps with about 20 Kbps standard deviation, whereas the corresponding rates for layers 5 and 11 were 250 Kbps (mean) and 130 (variance), and 20 Kbps (mean) and 15 (variance), respectively. With the average rate of the total output at 2.7 Mbps, subband 1 consisted of less than 25% of the total rate, which implies that users can participate in a session even if their access bandwidth is only a quarter of that needed for receiving a complete signal. This scheme provides very good error containment in case of lost packets; however, due to the low rate of some of the subbands, packetization delay is large for those subbands and essentially for the whole signal. Combining some of the lowest-rate subbands helps to reduce this delay.

It is interesting to note that the emerging standard for video compression, named *MPEG* [IEEE 1989], may also be viewed as hierarchically structured. Every eighth frame is a reference frame containing the complete set of parameters needed for frame reconstruction at the receiver, whereas other frames (the interframes) carry only information about changes from these reference frames. Receiving only the reference frames (which constitute the lowest layer) results in low-quality video, which improves as more interframes are received.

Hierarchical coding provides the basis to which the source, destinations, and networks specify requirements and set call-level parameters. Destinations can declare signal-delivery parameters, such as desired bandwidth; and the network uses end-users' requirements and the source's signal structure to compute routes to all destinations. Furthermore, the source can base the number of signal layers and the bandwidth assigned to each layer on the availability of network bandwidth and on the destinations' requirements, so as to maximize the utility of the signal delivered to the destinations. These aspects of hierarchical encoding are discussed in the next section.

## 3.2 SETTING CALL-LEVEL PARAMETERS

The processor responsible for setting up a hierarchical multicast session must first receive as an input the destinations' requirements. Based on this input, and the network conditions, the processor determines the major parameters of the session by the following steps:

1. Computing the bandwidth available to each destination. The most important factor is the maximum signal bandwidth that can be delivered to each destination, given the smallest of (1) destination's maximum bandwidth requirement and (2) the maximum path bandwidth available to that destination. Other constraints that may be taken into consideration include the minimum-quality signal accepted by each destination and the penalty for failing to deliver the minimum-quality signal to a given destination.
2. Setting parameters of the signal layer structure to maximize the overall quality delivered to the destinations. By this we mean assigning bandwidth to each layer under constraints such as a fixed number of layers and constant overall signal bandwidth.

3. Computing a set of paths to deliver to each destination the maximum possible number of layers. The paths must result in efficient network utilization, where no link carries more traffic than is actually delivered to the destinations on the paths of which that link is a part. The paths must also be optimal with respect to user requirements or cost. For example, if delay is a critical factor, the shortest path with maximum bandwidth should be used.

In the rest of this section we present algorithms for accomplishing these tasks, under the constraint that all the layers delivered to a given destination follow the same path to that destination. First, however, we describe the network and coder models.

### 3.2.1 The Mode

The network is modeled by a graph $G = (V, E)$, where $V$ and $E$ are the sets of nodes and links, respectively. Each link $(i, j) \subset E$ is characterized by its available capacity $b_{i,j}$. The available capacity of the path $\{i_1, i_2, \ldots, i_n\}$ is defined as $\min_j \{b_{i_j, i_{j+1}}\}$, $1 \le j \le n-1$.

To extend the model to also incorporate the destination's bandwidth requirements, the graph $G$ is augmented by $N$ links and $N$ nodes, where $N$ is the number of multicast destinations. For each destination $d$, which is connected to node $e$ of the original graph and requires signal at bandwidth $W_d$, a node $d$ and a link $(e, d)$ are added with link capacity $b_{e,d} = W_d$.

We follow the link-state approach to route computation by assuming that the full network topology and bandwidth availability are known to the processing units that compute routes.

The source generates its signal in a hierarchy of up to $K$ layers, where layer 1 (the lowest layer) represents the basic signal and Layer $i$, $(1 < i \le K)$ improves upon the quality of the signal constructed from layers $1, 2, \ldots, i-1$. We assume that the maximum number of layers, $K$, and the total bandwidth of the full signal are both fixed, but that the bandwidth assigned to each layer can be modified by the source. We denote the bandwidth assigned to the $i$-th layer by $LB_i$. Thus, to receive the signal at level $i$, a destination must have access bandwidth of at least $W_i = \sum_{j=1}^{i} LB_j$.

### 3.2.2 Maximum Bandwidth Computation

Given the aforementioned model, we use a Dijkstra-like algorithm to compute the maximum single-path bandwidth to all destinations. The algorithm begins by labeling the source, say node 1, by $B_1 = \infty$, and all other nodes are temporarily labeled by zero, i.e., $B_i' = 0$. The label values represent the maximum path capacity from the source to each of the nodes. The algorithm's first step is to assign to all of node 1's neighbor nodes temporary labels, $B_i'$, according to $B_i' = b_{1,i}$. The node with the maximum label value, say node 2, is assigned a permanent label $B_2 = B_2'$. The temporary labels of node 2's neighbor nodes are then modified according to

$$B_i' \equiv \max(B_i', \min\{B_i', b_{2,i}\}) \qquad (11)$$

That is, the capacity of the path from node 1 to $i$ through 2 is the maximum between its previous label and the path capacity through node 2. After this temporary labeling, the node with the maximum temporary label is permanently labeled (i.e., converting $B_i'$ to $B_i$). The process repeats itself, where in a typical step the node with the largest temporary label is permanently labeled, and the temporary labels of all its neighbors are accordingly modified. The algorithm terminates when the last node of the graph is permanently labeled. It can be shown that a node's permanent label so assigned has a value that equals the maximum capacity of all paths from the source to that node.

It is interesting to note that this algorithm assigns permanent label values in a nondecreasing sequence. Consequently, once the algorithm computes permanent labels for all the multicast destination nodes and those nodes with labels not strictly smaller than the smallest destination label, there is no path through unassigned nodes that can carry enough layers of the signal hierarchy needed for a destination.

This algorithm provides information about the bandwidth available to all destinations. We will see in Subsection 3.2.4 how to use this information to compute maximum-capacity, shortest-path routes to all destinations. But first we discuss how the source can use this information to optimize its signal structure.

### 3.2.3 Optimization of Signal Structure

The maximum-bandwidth algorithm returns a list of destinations and the bandwidth they can receive. If the signal layer structure is already determined, this information can be translated into the number of layers to each destination. However, when the signal structure is flexible, the bandwidth occupied by each layer can be set to maximize the quality seen by the set of destinations in the specific session.

Suppose there are $N$ destinations and they can receive the signal in $L$ different bandwidth levels. That is, $n_i$ destinations can receive the signal at bandwidth no larger than $W_i$, where $i = 1, 2, \ldots, L$ and $\sum_{i=1}^{L} n_i = N$. Suppose that a measure of signal quality is assigned to each bandwidth: that is, if a destination receives the signal at bandwidth $W_i$, the quality of the received signal is $q_i > 0$, where $q_i > q_j$ if $W_i > W_j$. Notice that a destination with available bandwidth $W_i$ may receive the signal at a lower bandwidth level, in which case the received quality is according to the bandwidth actually delivered.

The source is assumed to generate its signal in up to $K$ layers. If $K \geq L$, the source can send its signal in $L$ layers corresponding to the bandwidth levels available to the destinations, thereby matching the demands of all destinations. The more interesting, and common, case is when $K < L$. In this case, the source selects $K$ bandwidth levels $W_{i_1}, W_{i_2}, \ldots, W_{i_k}$, and encodes its signal at these levels. That is, the first layer is assigned bandwidth $W_{i_1}$, the second $W_{i_2} - W_{i_1}$, etc., where

$W_{i_K} \geq W_L$. Notice that all destinations at level $W_j$, $(W_{i_k} \leq W_j < W_{i_{k+1}})$ receive the signal at the bandwidth $W_{i_k}$, and the corresponding quality $q_{i_k}$. Given this choice of signal levels, the average signal quality received by the destination population is defined as

$$\begin{aligned} Q_K^L(i_1, i_2, \ldots, i_K) &= (n_{i_1} + \ldots n_{i_2 - 1}) q_{i_1} \\ &\quad + (n_{i_j} + \ldots n_{i_{j+1}} - 1) q_{i_1} \\ &\quad + (n_{i_K} + \ldots n_{i_L}) q_{i_1} \end{aligned} \tag{12}$$

In this equation, it is assumed that destination $j$ does not receive a signal at all if $W_j < W_1$, in which case $q_j = 0$. We formulate the signal design problem as follows: based on the knowledge of the bandwidth levels available to the destinations, assign bandwidth to the $K$ layers so that the total received quality is maximized. That is, the source attempts to achieve a total quality of

$$\begin{aligned} Q_K^L = \max_{1 \leq i_1 \leq \ldots \leq i_K \leq L} & ((n_{i_1} + \ldots n_{i_2 - 1}) q_{i_1} \\ & + \ldots + (n_{i_j} + \ldots n_{i_{j+1}} - 1) q_{i_1} + \ldots \\ & + (n_{i_K} + \ldots n_{i_L}) q_{i_1}) \end{aligned} \tag{13}$$

This problem can be solved by an iterative procedure, which we present below. Notice that for $K = 1$,

$$Q_1^L = \max_{i_1} [(n_{i_1} + \ldots + n_L) q_{i_1}] \tag{14}$$

We denote

$$Q_1(i,j) = (n_i + \ldots + n_j) q_i \tag{15}$$

implying that

$$Q_1^L = \max_i [Q_1(i, L)] \tag{16}$$

For $k \geq 1$ we define

$$Q_k(i_1, L) = \max_{i_2 \leq \ldots \leq i_K \leq L} [Q_1(i_1, i_2 - 1) + Q_1(i_2, i_3 - 1) + \ldots + Q_1(i_k, L)] \tag{17}$$

That is, $Q_k(i, L)$ is the maximum average quality for a $k$ level signal, given that its first level is $i_1$.

Using the notation above we can write:

$$\begin{aligned} Q_k(i_1, L) &= \max_{i_2 \le \dots \le i_K \le L} [Q_1(i_1, i_2 - 1) + Q_1(i_2, i_3 - 1) + \dots + Q_1(i_k, L)] \\ &= \max_{i_2} [Q_1(i_1, i_2 - 1)] + \max_{i_3 \le \dots \le i_K \le L} [Q_1(i_2, i_3 - 1) + \dots + Q_1(i_k, L)] \\ &= \max_{i_2} [Q_1(i_1, i_2 - 1) + Q_{K-1}(i_2, L)] \end{aligned} \tag{18}$$

The maximum value is given by

$$\begin{aligned} Q_K^L &= \max_{i_1 \le \dots \le i_K \le L} [Q_1(i_1, i_2 - 1) + Q_1(i_2, i_3 - 1) + \dots + Q_1(i_k, L)] \\ &= \max_{i_2} [\max_{i_1} [Q_1(i_1, i_2 - 1)] + \max_{i_3 \le \dots \le i_K \le L} [Q_1(i_2, i_3 - 1) + \dots + Q_1(i_k, L)]] \\ &= \max_{i_2} \left[Q_1^{i_2 - 1} + Q_{K-1}(i_2, L)\right] \end{aligned} \tag{19}$$

The recursive expression for $Q_K(i_1, L)$ can be used to compute these values by first computing the $\binom{L}{2}$ values for $Q_1(i, j)$, $1 \le i < j \le L$, and then, for $2 \le k < L$ and $2 \le i \le L - K + 1$, computing $Q_k(i, L)$. The maximum value, $Q_K^L$, is computed using equation 19.

Although this iterative solution method is demonstrated above only for the case in which maximum bandwidth is specified for each destination, the same method can accommodate additional types of constraints. Such constraints are, for example, a requirement that all destinations to receive at least the lowest layer of the signal and a specification of the minimum acceptable signal bandwidth for each destination.

## 3.2.4 Computation of a Multicast Tree

The two algorithms specified above result in signal structure that is optimally tuned for the specific set of destinations and network conditions. Furthermore, the first algorithm, which computes the maximum bandwidth available to all destination, also provides paths with the corresponding bandwidth. When that algorithm terminates, a node's label is equal to the maximum bandwidth path leading to that destination. The set of links connecting nodes to the nodes they label is a tree by construction, and the paths along this tree possess the maximum bandwidth available to each node on them.

Note, however, that constructing this tree is suboptimal for routing packets, for the following reasons:

- There may be shorter paths with the same bandwidth, which would be more desirable for packet routing.
- The node labels on the tree do not necessarily represent the actual utilization of the tree link.

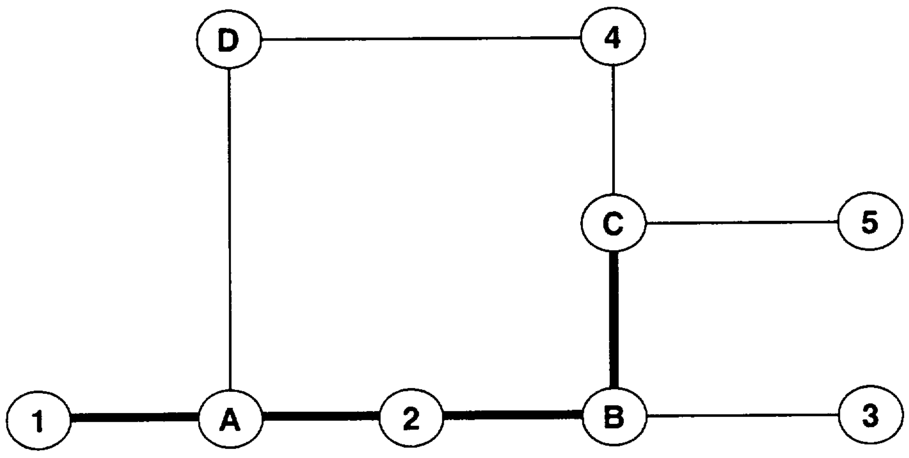
For an example of the latter, consider Figure 8, where the source is node 1 and the destinations are 2, 3, 4, and 5. The thin and thick lines represent link bandwidths sufficient for carrying signal level 1 only and the whole signal, respectively. The maximum bandwidth spanning tree, which consists of all the links except (A,D) and (D,4), indicates that destination 2 can receive the full signal, whereas 3, 4, and 5 can receive only level 1.

Using the tree as computed implies that node 1 sends the full signal along the tree path to node B, which replicates level 1 for forwarding to 3, and forwards the complete signal to C, which forwards only level 1 to destinations 4 and 5. We denote nodes B and C as *filters*, since some of their output carry fewer signal layers than those arriving at the node's input. Using the tree as described is inefficient. First, link (B,C) should carry only layer 1 of the signal, since only this layer reaches the destinations (4 and 5) to which this link leads. The same argument applies to link (2,B). Reducing the signal level carried on these links shifts the filter role from B and C to node 2. That is, the complete signal is carried only on links (1,A) and (A,2), and on the rest only layer 1 is carried. However, with this new bandwidth assignment, it is preferable to send traffic to node 4 through links (A,D) and (D,4), which are not even on the maximum bandwidth tree, since this path has the same bandwidth as the tree path but is shorter.

As illustrated in Figure 8, the maximum-bandwidth tree algorithm has two major deficiencies:

- It does not guarantee shortest paths.
- It includes wideband links leading to destinations that can receive only lower bandwidths.

Some improvements in the algorithm can be made, as follows.



| LEGEND | |
|---|---|
| 1 | Source |
| 2, 3, 4, 5 | Destinations |
| A, B, C, D | Nodes |

**Figure 8. Multicast Routing with Two Link Types**

Shorter paths can be obtained by selecting the node with the shortest distance to the source during the execution of the algorithm in the step where a node is selected for inclusion in P, if there are several nodes with maximum value in T. For example, in Figure 8 node D should be chosen as a predecessor for node 4 since it is closer to the source than C.

After the termination of the algorithm, bandwidth allocation to links that are incorporated with too high a bandwidth (such as link (B,C) in Figure 8) can be reduced by the following procedure. A destination is selected that has an assigned bandwidth lower than the maximum (say, node 3). Its tree path is followed to the source, and any link bandwidth is reduced to the destination's bandwidth. This operation continues until the path meets a node with an outgoing link of higher bandwidth than that of the selected destination (node B). At that point, another destination is selected (say, node 5) and the procedure continues (this time reducing the bandwidth assignment of [B,C] and [2,B]). Notice, however, that since the assignment of the paths is based on maximum bandwidth, this "bandwidth trimming" procedure may still result in some paths that are longer than others with the same bandwidth. For example, the route to node 4 does not carry more traffic than needed, but is not the shortest.

To remedy these deficiencies, we modified the above algorithm so that it incorporates bandwidth trimming more often. This is done by first considering the links with maximum available bandwidth W_K, and then constructing a shortest-path spanning tree using only these links. If this spanning tree reaches all destinations of the multicast session, the procedure ends with all destinations capable of receiving the full signal. Otherwise, the tree must be expanded with links of $W_{K-1}$ to additional destinations that can receive only K-1 levels of the source signal. For example, in Figure 8 this first step produces a tree with nodes 1, A, 2, B, C, and the thick links connecting them.

Notice that the subtree already constructed using the links with $W_K$ may contain overutilized links. These are the links that do not lead to a destination after the first step; therefore, the path to any destination they may lead to in future steps must contain a lower-bandwidth link, thereby restricting the path's bandwidth below $W_K$. These overutilized links are eliminated from the highest-bandwidth spanning tree by selecting a tree leaf that is not a multicast destination, and deleting the tree link leading to it. The operation is repeated until there are no leaves that are not destinations. The resulting tree is the basis for expansion. In our example, nodes C and B are not destinations; therefore, links (2,B) and (B,C) are deleted from the tree.

The tree is expanded by the use of all the links with available bandwidth W_K that are not included in the tree and the links with $W_{K-1}$, with all these links considered as having an available bandwidth of $W_{K-1}$. Via these links, paths with shortest distance from the source are obtained to all the nodes reachable by this additional set of links. Again, in our example, links (2,B) and (B,C) are now becoming thin links and are used to expand the tree along with the other thin links. The links that do not lead to a destination are eliminated, and their capacities are reassessed as having been "reduced" to $W_{K-2}$. The algorithm continues in this manner until all the multicast destinations are reached.

The tree expansion procedure described above involves an important tradeoff. Recall that the input to the first expansion is the set of links with $W_{K-1}$ and the $W_K$ spanning tree with minimum distance assigned to each of its nodes. In the expansion, we either do or do not allow changes in the distances of the first tree. A node of the $W_K$ tree may change its distance when a link of $W_{K-1}$

completes a cycle. Since we do not want to eliminate the $W_K$ links in the first tree (since they provide the broadband path to that node), we may either allow the $W_{K-1}$ link to be added, thereby destroying the tree structure, or we may not allow the inclusion of those links, in which case the resulting paths to the $W_{K-1}$ nodes may not be the shortest. As an example of such a situation, consider the graph in Figure 9, in which nodes 2 and 3 are the destinations. The path to node 2 is 1,A,B,C,D,2, on which the complete signal is delivered. When extending the path to node 3 we face the following situation: using the previous path and extending it by (2,3) preserves the tree structure. However, the path 1,A,E,2,3 is shorter yet delivers the same bandwidth to node 3; therefore it is preferable to the other path, but it destroys the tree structure of the paths.

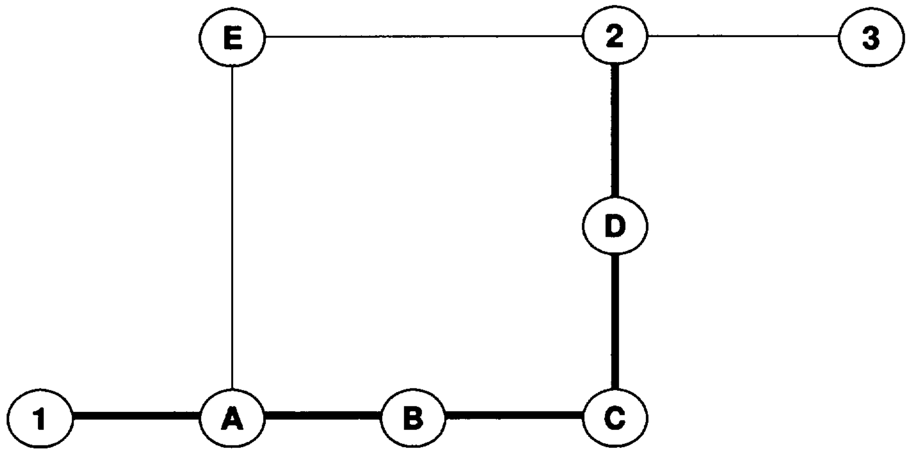


**Figure 9. Maximum Bandwidth Spanning Tree**

## 3.3 ERROR-CONTROL PROCEDURES

In the previous section we presented procedures for setting session parameters such as destination requirements, signal structure, and maximum-bandwidth paths. During a session, the source sends its signal in packets, each of which carries bits of a single layer, to allow easy filtering of layers in the network. In this section we discuss issues in maintaining the integrity of such a layered packet stream.

Packet transport in hierarchical multicast is different from that in a "regular" session: in a regular, unicast, or multicast session, all packets form a single stream in which all packets are equally important and each has to reach all destinations. In hierarchical multicast, on the other hand, different layers require different degrees of protection, and different destinations receive different subsets of the signal layers. These differences result in requirements for special error-control procedures, as discussed below.

Since broadband signals used in hierarchical multicast are likely to be distributed in a high-speed, fiber-optic-based network, data distortion is expected to be dominated by packet loss due to congestion and admission control, rather than by noise-inflicted bit errors. For this reason, the focus here is on techniques for the recovery of lost packets rather than the correction of bit errors. Furthermore, the focus here is on FEC rather than on retransmission-based techniques such as ARQ [Burton and Sullivan 1972; Schwartz 1987] because of the real-time delivery requirements, which are likely to be imposed on traffic amenable to hierarchical coding.

FEC-based packet recovery schemes make use of the fact that recipients can identify the location of missing packets by the gaps in the sequence numbers of the arriving stream. This fact allows the recipient to treat a lost packet as a sequence of *erasures* rather than a burst of errors, thereby requiring much less overhead for recovery.

When designing an error-control technique for packet recovery, one has first to recognize that carrying control bits in the same packet with the data bits they are to protect is not very useful, since the error-control bits are also lost when the packet is discarded. Rather, we propose the use of error-control schemes that (1) protect a group of packets at a time, (2) provide error-control redundancy for the whole group, and (3) carry the control bits separately from the data, so that losing a packet affects either data or control bits but not both. Carrying error-control and data bits in separate packets is also useful in cases such as video coding, where the source emits its data in packet-ready format without including error control. Adding error-control packets is easier in these cases than modifying the packet structure to add error-control bits, since the latter often require modification of the source, whereas the former do not.

A basic FEC scheme described by Shacham and McKenney [1990] groups the source's packets into L packet blocks, and adds to each block *parity packets*, where each bit in a parity packet is a function of the corresponding bits in the data packets of the block. For example, the *i*-th bit of a parity packet may be the modulo 2 sum of the *i*-th bits of the data packets. Additional parity packets can be constructed via orthogonal sets of equations. The advantage of this scheme is that parity packets can be constructed "on the fly" by very simple hardware, without delaying the transmission of the first data packet until the last packet of the block passes through the coder, so that there is no need to wait for the arrival of the whole block before decoding begins. Decoding is also very simple and has the additional advantage that packets need not be decoded in the order in which they pass through the coder. However, if the packet stream must be forwarded to the recipient in order, blocks with missing packets must be delayed until the last packet arrives, so that the recovered packet(s) can be inserted in the correct place in the sequence. Another advantage of this scheme is that it generates a systematic code that leaves the data packets intact and is usable when other packets are lost and cannot be recovered.

The performance of this erasure recovery scheme depends on the amount of parity packets that are added to the data and on the distribution of packet losses. Since packets are lost because of excess traffic, adding parity packets increases the traffic and thereby increases the packet loss rate. On the other hand, adding too few parity packets makes the erasure recovery technique too weak. It was found that about 5% of parity information is sufficient to reduce the packet loss rate by two to three orders of magnitude, when the added loss rate due to traffic increase was taken into consideration.

This impressive gain, however, was computed on the assumption that packets are lost independently. Correlation in the packet loss process, i.e., when packets are lost in bursts, reduces the performance of this scheme. It is known that the buffer overflow process is bursty: the conditional probability that an arriving packet will find the buffer full is higher if the previous packet found it full than if the previous packet found that the buffer had storage space. Recovering bursts of lost packets requires more overhead and more complex decoding schemes than is required for the recovery of packets that are lost independently. Moreover, bursts of errors cause more visible damage than randomly generated errors. Thus, the performance of FEC can be greatly improved by provisions for dispersing bursts of lost packets.

Notice that packets are discarded by network nodes according to a human-designed algorithm, as opposed to the "natural" noise process that causes transmission errors. The total number of packets that are discarded is a result of overall congestion and may not be easy to adjust in real time. However, which packets are being discarded and which are kept in the buffer can be easily affected by changing the rules that govern admission to the buffer and order of service (priority) of the packets. This implies that buffer management procedures can help improve conditions for error control.

These considerations are true for any type of traffic, but they have special implications for hierarchical data. First, the complete stream issued by the source may not arrive intact at each destination. In fact, as discussed above, the main reason for employing hierarchical data is to allow some layers to be "peeled off" inside the network. Consequently, if packets of different layers are combined in the same FEC block, intentional discarding of packets from one layer severely affects the ability to recover packets from another layer that are lost due to buffer overflow. The objective, however, should be that the ability to recover lost packets of (for instance) layer 1 is independent of the arrival of higher layers.

To achieve this goal, the erasure recovery scheme is augmented by applying parity packets to blocks consisting of data packets of the same layer. The advantages of this provision are as follows:

- Layers can be coded/decoded independently.
- Different erasure recovery codes or rates can be applied to different layers, to match the level of protection to the signal quality deterioration due to packet loss at the various layers.
- Parity packets can be added only to part of the traffic, such as the lowest layer, resulting in a lower increase in traffic (hence in loss rate) than if the whole stream were encoded.

A disadvantage of this separation is that it requires as many coders and decoders as there are protected layers, whereas only one coder and decoder are needed per source/destination when the whole stream is encoded together. An additional disadvantage is that more delay is needed to resequence a recovered stream if it is layered than if it is a single stream. However, the advantages above clearly outweigh the disadvantages.

Techniques for dispersing lost packets are of special importance for hierarchical data. For example, a burst of lost packets at the lowest layer of a video signal can cause clearly visible degradation of the image for several seconds. There are two basic mechanisms for alleviating the effect of bursty losses in the network:

- Priority-based discarding of packets
- Arranging the source's packet stream as a rectangular array, and providing parity packets to groups of data packets constituting both rows and columns of such an array.

When encoding is applied to an entire stream, buffer management procedures disperse packet loss by discarding packets based on their FEC block affiliation and on the number of packets already lost in their respective blocks. For example, if the code can recover only one packet per block, discarding two packets from a block prevents the recovery of both packets. Thus, if the second packet is about to be discarded from a block, the buffer server selects a packet from another block that has not yet lost any packet and discards it instead, thereby making both packets recoverable.

With hierarchical data and separate layer encoding, the buffer has an additional degree of freedom in that it can choose packets for discarding from different layers. Since the various layers are interleaved, there is a higher probability of finding a substitute packet that prevents a second packet loss in a block of a given layer. The most natural choice is to discard higher-layer packets before lower-layer packets are discarded. Notice that in some hierarchical representations, once a packet is discarded, all packets that contain the higher-layer information of that packet lose their value and therefore may also be discarded. The aforementioned layer-based priorities are for packet discarding and do not imply order of service. Since all packets must be delivered below some given delay, high-layer packets are served before low-layer packets that represent later segments of the signal.

The interleaving of the layers provides a natural dispersion of bursts of lost packets for any specific layer. For example, if the lowest-layer packets represent $1/N$-th of the total stream, the number of packets lost from this layer during a burst is that fraction of the length of the burst.

Notice that both the layer-based discarding priority and the burst dispersion have maximum effect where the whole source stream is present, say, at node D in Figure 9. However, as high layers are removed due to lack of bandwidth, the effect of this "cushioning" for the low-layer packets diminishes. Over narrow-band links, after high layers have been removed, the low-layer packets have no other packets to preempt, which is likely to increase the loss rate of the low-layer packets and the burstiness of that loss process.

In cases where a significant portion of the traffic goes through narrow-band paths, the source should employ the other loss dispersion mechanism mentioned above, namely, interleaved coding.

## 3.4 CONCLUSION

We have introduced a multipoint communication paradigm that combines hierarchical coding of the source's signal with the delivery of different subsets of layers to destinations based on their terminal and access constraints. This paradigm facilitates the distribution of broadband information in heterogeneous environments and allows more destinations to participate in a multicast session, despite differences in the rates at which they can receive or prefer to receive information.

Hierarchical multicast requires new protocol functionality at the source, the destination, and the data distribution network. This paper has discussed some of the major areas that are essential to such operation:

- Hierarchical signal representation
- Computation of the maximum bandwidth available to each user
- Assignment of bandwidth to the signal layer to maximize overall received quality
- Determining the shortest-path tree whose paths are of maximum bandwidth
- Efficient assignment of traffic volume to links
- Error-control procedures aimed at recovering lost packets.

The maximum-bandwidth algorithm, signal optimization procedure, tree algorithm, and traffic assignment are aimed at static conditions and are therefore applicable to long-term actions such as call setup. These techniques must be further enhanced to operate in dynamic network conditions that arise from traffic variations and changes in destination population.

Note also that these techniques were developed under the assumption that all the signal layers are carried on the same path from the source to a specific destination. There are certain benefits to such a restriction, including the flexibility to protect lower-layer packets by the deletion of higher-layer packets, reduction of delay jitter at the receiver, and simplification of session management. Allowing traffic to be carried to a destination over multiple paths can yield more bandwidth, hence higher quality, to individual destinations.

In addition, the above mechanisms should be supplemented by admission control and resource reservation for the various signal layers. These mechanisms must take into consideration the difference in importance among the layers and must provide the smoothest possible delivery for the lowest layer—if necessary, at the expense of higher layers.

# 4 AN ALGORITHM FOR OPTIMAL MULTICAST OF MULTIMEDIA STREAMS

Multimedia communication that integrates voice and/or data and human and/or computer-generated information is likely to be the cornerstone of many future applications: it is expected to greatly enhance the versatility and effectiveness of human information exchange (teleconferencing) and human-machine interaction [Jayant 1992]. In a multimedia session, a source typically generates multiple streams, each representing a different medium, with possibly multiple video images, graphics, and audio. The high bandwidth and processing requirements of such sessions has thus far impeded their implementation, due to the limitations of traditional networking technology. Recent advances, however, in processing, transmission, and switching technologies have provided the physical infrastructure needed to support broadband, multimedia, multiparty information exchange. Other essential ingredients, like routing, traffic control, and resource allocation protocols, are still active areas of research.

An important requirement of traffic-control mechanisms for real-time multicast is a means of distributing multiple streams from source(s) to a set of destinations. Certain network technologies (mostly local-area networks) support traffic distribution to all connected users at the physical level, most notably by a common bus or ring to which all the users have access. This task is significantly more difficult in networks with mesh topologies, which are required to employ multicast routing and transport protocols to effectively deliver the transmitted streams to the requesting destinations without unnecessary consumption of network resources. Mesh topologies used to be typical in wide-area networks, while LANs employed specially configured topologies such as bus or ring. Recently, however, mesh topologies have also been employed by high-speed LANs as a means of increasing network size and total throughput. Thus, multiparty traffic distribution mechanisms will be required for both LANs and WANs.

Existing multicast protocols approach the distribution task by attempting to deliver all generated streams to all destinations. This approach is intuitively appealing, because if all destinations receive the same information, all can communicate on an equal basis. In many cases, however, this approach may be infeasible, inefficient, too constraining, or simply undesirable. For example, when the total bandwidth required by all streams exceeds the capacity of the network, the network is simply unable to deliver all streams. Such limitations are likely to be particularly important in real-time video-based communication, in which the number of multimedia streams being offered to the network increases with the number of participants. Consequently, the number of participants and variety of data streams that can be exchanged among them may be severely constrained.

The limitations of the traditional approach are particularly restrictive in *heterogeneous* networks, where the links have different capacities and terminal equipment capabilities vary considerably among the destinations [Ballardie, Francis, and Crowcroft 1993]. Moreover, the cost of participating in a session increases with the delivered data rate; therefore, a user may prefer to limit the scope of his/her participation by receiving only a subset of the generated traffic, e.g., by electing to view only some of the offered images at any given time. Furthermore, in a multiparty session, users are likely to differ in the streams they wish to receive at any given time, thereby forcing the network to deliver different streams to different users, which gives rise to another form of heterogeneity.

The effect of bandwidth constraints is to limit the subsets of generated streams that can be delivered to destinations, whereas heterogeneity imposes different limits and constraints across the network. In both cases, attempting to deliver all generated streams to all users may result in severe inefficiencies and uncontrolled packet losses, which in turn degrades the quality of the received streams. Furthermore, in heterogeneous environments the received quality will vary among users. Attempting to make the received quality equal while insisting on uniform delivery results in a dilemma: if all users who wish to participate are allowed to be in the session regardless of their constraints, the quality of the session is driven by the ability of the least capable destination. On the other hand, if the more capable users insist on a certain minimum quality, destinations that cannot participate at that level must be excluded.

We propose here a way around this dilemma in which the traffic distribution algorithm takes into consideration all user requests and network bandwidth constraints, and delivers to each user a subset of streams according to the user's individual needs and capabilities. That is, a stream is distributed not necessarily to all destinations, but only to those who request it and are capable of receiving it. Other streams may be distributed to different groups of destinations. Such a scheme, which we call ***heterogenous multicast,*** was presented by Shacham [1992] in the context of the distribution of a single stream that is encoded in layers (also known as *hierarchically encoded*) [Karlsson and Vetterli 1989; Ghanbari 1989]. The protocol delivers to destination $j$ the layers $1, 2, \ldots, l_j$, where $l_j$ is determined by the maximum-capacity path to $j$.

In this section, we generalize the work by Shacham [1992] to the case of a single-source multicast of multiple streams, some of which may be hierarchically encoded. Each destination specifies the subset of streams and layers it wishes to receive, and the distribution algorithm presented in this paper attempts to satisfy all individual demands. The main difference between this paper and the work in the hierarchical model presented by Shacham is that in the method we propose, a destination can specify and receive a much wider range of subsets of streams than the ordered set $1, 2, \ldots, l$ allowed by Shacham. The relaxation of subset selection requires a new method for stream selection and distribution.

To deliver the streams to their destinations, a multicast tree from the source to all destinations is established as a part of the session setup procedure, with all streams from the source to all destinations being candidates for transport on that tree. A single multicast tree for all streams is advantageous in multimedia communications because it facilitates media synchronization and simplifies traffic management. So that the largest number of streams can be carried, it is assumed here, as it was by Shacham, that a *maximum-bandwidth* tree, which consists of the highest-capacity paths from the source to each destination, is established at the beginning of the session. Such a tree is depicted in Figure 10, where the link capacities are marked in angular brackets; the destinations are nodes A, B, and C; and S is the source.

The task we face is that of providing a unidirectional distribution of multiple streams, with different bandwidth requirements, over a tree with heterogeneous link capacities to a set of destinations, each of which places an individual demand for a subset of the generated streams.

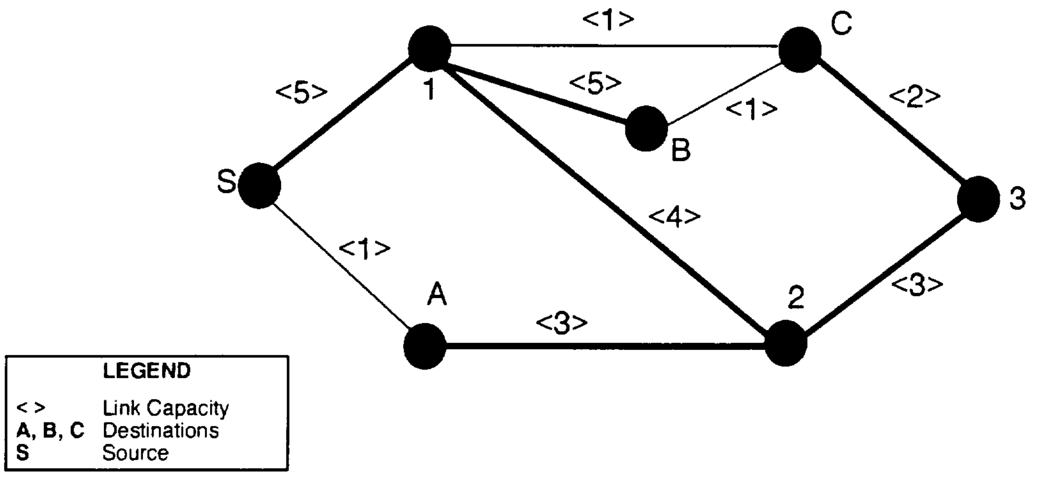


**Figure 10. Maximum-Bandwidth Tree**

When all demands cannot be satisfied in this heterogeneous environment, a problem arises as to how to select and distribute the "best" sets of streams. There are clearly many possibilities for stream distributions, where each distribution amounts to assigning streams to tree links.

A natural question that arises in this situation is how to compare different distributions. Once a suitable metric and a selection criterion are decided upon, an algorithm can be developed to compute the optimal distribution. To this end, we find the concept of a *bid* to be very effective. A bid is a nonnegative numerical value assigned by each destination to each stream, and represents the level of priority assigned by the destination to the reception of that particular stream.* For every stream delivered on a tree leaf, the source gains an amount equal to the sum of the bids for that stream assigned by the destinations on that leaf. The total gain for a given distribution is defined as the sum of all the bids for all streams over all leaves to which the streams are delivered. Stream distributions can thus be compared on the basis of their total gain; the objective of such a comparison is to select a distribution that maximizes the total gain to the source.

Extensive research has been conducted on multicast traffic distribution, with most of the attention given to finding distribution (spanning) trees under various conditions and with various objectives, including minimizing the total cost of the tree or the average delay to the users [Bharath-Kumar and Jaffe 1983], limited network status information (e.g., [Ballardie, Francis, and Crowcroft 1973; Bharath-Kumar and Jaffe 1983]), and flexibility in modifying the tree during the session [Waxman 1993; Doar and Leslie 1993]. The effect of streams on network capacity and the need to select stream subsets has only been mentioned in passing [Waxman 1988], but no algorithm or analysis has been provided. In the following subsections we address this need.

*Bids are discussed in detail in Subsection 2.3.

## 4.1 THE STREAM-SELECTION PROBLEM

This discussion is structured as follows: In Subsection 4.1 we formulate the stream distribution problem as one of mathematical optimization; in Subsection 4.2 we develop an algorithm to compute the optimal distribution. In Subsection 4.4 we prove the algorithm's correctness and in Subsection 4.5 we ascertain its complexity. Subsection 4.6 contains some concluding remarks.

### 4.1.1 Bandwidth Constraints

In the model considered here, a multimedia source $S$ is connected to $M$ destinations $(1,\ldots,M)$ by a multicast tree $G(V,E,C)$ where $V$, $E$, and $C$ represent network nodes, links, and link capacities, respectively. An example is shown in Figure 11, where the link capacities are denoted by $\langle C_j \rangle$, $j = 1, \ldots, L$. The source, which is connected to the distribution tree through the access link $s$, generates $N$ real-time streams where stream $i$ requires bandwidth $w_i$, $i = 1, \ldots, N$. It should be noted that assuming a single access link for the source does not restrict the generality of our results. Specifically, if the source is connected to multiple subtrees by separate access links, the streams on the different subtrees do not compete for bandwidth with one another. Hence, individual subtrees (rooted at the source) can be studied separately.

Given a maximum-bandwidth tree, the capacity of the path from the source to destination $i$ is defined as $\min_j \{ C_j \}$, where $C_j$ is the capacity of link $j$ on the path. For example, in Figure 11 the capacities of the paths from the source to B and D are 3 and 2, respectively.

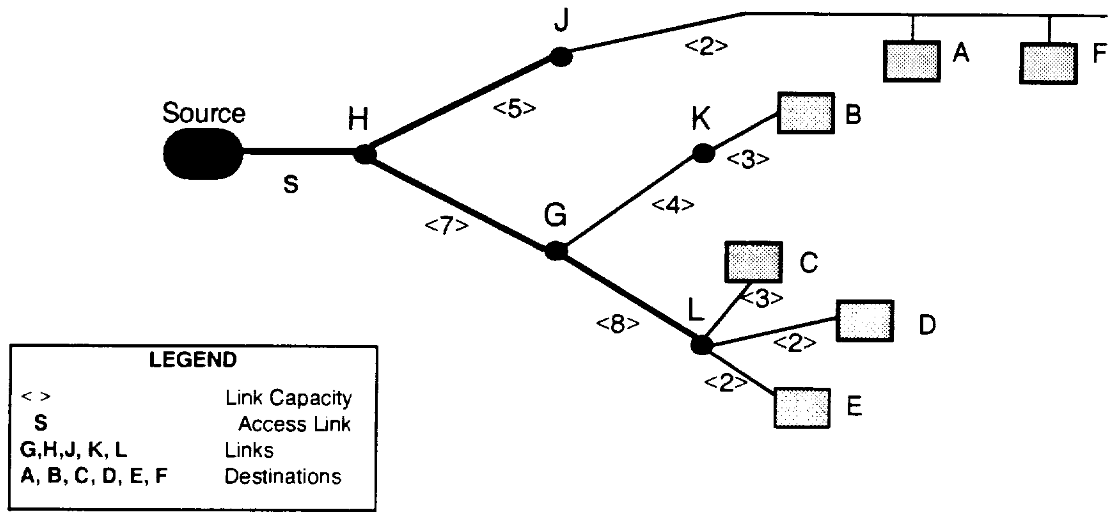


**Figure 11. Heterogeneous Multicast Tree**

### 4.1.2 Feasible Flows

We define a *flow* to be a set of streams and their assignment to links. It is convenient to represent a flow on the tree by means of the set of the stream assignment variables $\{X_{i,j}\}$, which are defined by

$$X_{i,j} = \begin{cases} 1 & \text{if stream } i \text{ is assigned to link } j \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

To be *feasible*, a flow must satisfy the following constraints:

1. **Continuity**. All flows originate at the source and no streams are generated at the tree nodes: i.e.,

$$X_{i,j} \leq X_{i,k} \tag{21}$$

where $j$ is down the tree from $k$.

2. **Link Bandwidth**. Streams are assigned to links only if the link bandwidth can support them; i.e., for all links $j$,

$$\sum_i X_{i,j} w_i \leq C_j \tag{22}$$

Notice that if $\sum_{i=1}^{N} w_i \leq C_j$, for all $j$, then the tree has sufficient bandwidth to carry all streams to all destinations; i.e., all flows that satisfy the continuity constraints are feasible. In this case, the multistream distribution task is straightforward: simply assign a stream only to the paths leading to the destinations with positive bids for that stream. However, if the bandwidth is limited and not all links can accommodate all streams, the stream distribution must be restricted by setting some of the $X_{i,j}$ values to zero, in spite of nonzero bids for them.

One can immediately see that even for a modest-size tree and a relatively small number of streams, the number of feasible flows can be very large. Of these feasible flows, we focus our interest on the flows that are

1. **Efficient**. A stream is not assigned to a nonleaf link if it is not assigned to at least one child branch of the tree. In terms of flow assignment variables, if $X_{i,j} = 1$, then $X_{i,k}$ cannot be zero for all children $k$ of link $j$.
2. **Undominated**. For feasible flows $X$ and $Y$, $Y$ dominates $X$ if $X_{i,j} \leq Y_{i,j}$ for all $i,j$ with at least one strict inequality. If flow $X$ is feasible and no feasible flow dominates $X$, $X$ is called undominated.

## 4.2 BIDS

A metric is needed to compare flows and select the "best" from the set of undominated flows (which can be large). Such a metric should reflect the degree to which the destinations' demands are satisfied and the network is utilized. To that end, participant $k$ assigns a nonnegative value, $B_{k,j} \geq 0$ (called a bid), to stream $j$. When a group of destinations share a leaf, which is the case when they are connected through a common-medium LAN, each stream delivered to the leaf is available to the whole group. Hence, instead of directly handling bids by individual destinations, it is more convenient in this case to deal with the bids presented to the network by the whole group on tree-leaf $j$. Consequently, we denote the bid on link (leaf) $i$ for stream $j$ by $b_{i,j} = \sum_{dest\ k \in leaf\ i} B_{k,j}$, with the summation being over all destinations connected to link $i$. The source is assumed to gain an amount $b_{i,j}$ when stream $j$ is delivered to the destination(s) on leaf $i$. If $b_{i,j} > 0$, but stream $j$ is not delivered to that leaf, there is no gain to the source.

The flow metric that we use is the sum of the bids gained by the flow for all streams and destinations. The objective of the selection algorithms is to find the flow that maximizes the total gain to the source.

### 4.2.1 Problem Formulation

Our problem can now be stated as follows: Find $LN$ numbers $X_{i,j}$ that take values in $\{0, 1\}$ and that maximize

$$\sum_{i,j} b_j X_{i,j} \qquad j \text{ is a leaf}$$

$$\text{subject to} \quad \sum_{i=1}^{N} w_i X_{i,j} \le C_j \qquad j \in T \tag{23}$$

$$X_{i,j} \le X_{i,l} \qquad j \text{ is a child of } l$$

This is a 0-1 integer programming problem, which is known to be NP-complete [Papadimitriu and Steiglitz 1982].

### 4.2.2 A Simple Case: Flow on a Single Link

To illustrate the difficulty of this problem, consider a simple case where the source and all destinations share a common-medium, multiple-access network, such as an Ethernet or a token ring. Every transmitted stream reaches all the destinations, and the medium bandwidth is assumed to be the bottleneck. In this case, the problem in Expression 4 is reduced to the knapsack problem [ibid]. The tree has only one link, so that the second subscript in the variables above is not needed. The variable $X_i$ now assumes a value of 1 if the source transmits stream $i$, and 0 otherwise. The corresponding optimization problem becomes that of maximizing

$$\sum_i b_i X_i \quad \text{subject to} \quad \sum_i w_i X_i \le C \quad . \tag{24}$$

In the next section, we present an algorithm for solving the problem in Expression 4.

## 4.3 THE OPTIMIZATION ALGORITHM

In principle, one can determine the best flow by listing all possible combinations of $X_{i,j}$ for $i = 0, 1, \ldots, N$ and $j = 1, 2, \ldots, L$. Each such binary vector, which has $NL$ components, is checked for feasibility, i.e., flow continuity and satisfaction of the bandwidth constraints. The feasible flows are ordered according to their gains; and from the flows that achieve the maximum gain, the one that minimizes $\sum X_{i,j}$ is selected. The last selection results in the highest gain per used network resource. This approach necessitates the evaluation of $2^{NL}$ vectors, which severely limits the "size" of the problems that are computationally tractable.

In the rest of this section we provide an alternative algorithm that reduces the number of examined vectors by progressing on the physical distribution tree from the source to the destinations, and identifying flows that are infeasible and inferior to ones that have already been found. Once identified, these flows are eliminated from further consideration down the tree, and flow feasibility is maintained as it progresses. The algorithm consists of three procedures:

1. **Link_feasible**, for finding all undominated (i.e., maximal) subsets over a single link
2. **Tree_feasible**, for using **Link_feasible** to produce all maximal feasible subsets over all the links of the physical multicast tree
3. **Value**, for evaluating the gain for each feasible maximal flow, i.e., the sum of bids for all the streams in the flow over all leaves to which the flow is distributed.

The algorithm starts from the tree root, which has an associated set of streams. For simplicity of presentation we assume that the root has only one child link. Notice that we can always convert a tree to this form by adding a node and connecting it to the root by a link with bandwidth equal to the maximum of the bandwidth of all child links of the root.

A more detailed explanation of the operation of these procedures is given below, along with semiformal descriptions.

### 4.3.1 A Procedure for Listing All Link_feasible Undominated Sets of Streams

The input to the procedure **Link_feasible** is the capacity of a link and the bandwidth requirements for *M* streams, which are available at the transmitting end of the link. The procedure's output is a list of all undominated subsets of streams. Notice that if a subset is undominated and feasible on a link, it is also maximal, in that no other stream can be added without violating the subset's feasibility.

**Link_feasible** examines subsets of the *M* streams, where each subset is represented by the M vectors $(b_1, \ldots, b_M)$, where $b_i = 1$ (0) if stream *i* belongs (does not belong) to the subset. The vectors are organized in *M* rows, where row *i* consists of vectors with a Hamming weight of *k*, i.e., $\sum_{i=1}^{M} b_i = k$. From each set in row *k* there are *k* pointers leading to the *k*-dominated vectors of weight $k-1$, denoted its children. Initially, all vectors are unmarked.

**Link_feasible** begins with row *M*; and if the subset with all streams is feasible, the procedure terminates. Otherwise, it continues to row $M-1$. Upon considering row *k*, a vector that is unmarked and is feasible is added to the output list and its children are marked as dominated. All children of a dominated vector are marked as dominated. **Link_feasible** terminates after the first row in which all vectors either are dominated or have been added to the output set. A semiformal description of the procedure is given below.

```
Link_feasible(link-bandwidth, vector)
Initialization:
    All vectors are unmarked
    All rows are initially marked "0"
    The set of undominated sets is empty
For all vectors in a row
    If a vector is marked dominated
```

```
        Mark all of its children vectors as dominated
        Else
        If the vector is infeasible
        mark row "1"
        Else
        add the vector to the list of undominated sets
        mark all of its children as dominated
    Upon completion of a row:
        If the row is marked "0"
        STOP. There are no more undominated sets to add
        Else
        Scan the next row.
```

### 4.3.2 A Procedure for Finding All Feasible Undominated Flows

The procedure **Tree_feasible** finds all undominated flows on the tree. Using **Link_feasible,** and starting from the link connecting the root to the tree, the procedure expands each undominated set on a link to its undominated subsets on the children links. **Tree_feasible** accepts as input the description of the physical multicast tree (topology, link bandwidths, and bids on the leaves) and the bandwidth requirements of the set of streams offered by the root. The procedure begins at the root and builds a extended tree as follows. In a typical step, **Tree_feasible** considers a link and a set that is available at the parent link. For each undominated subset (found by **Link_feasible**), the procedure appends all the children of that link to the extended tree and marks that undominated subset as available for those children. Another way of looking at this procedure is to see that **Tree_feasible** creates, for each undominated set found on a link, a replica of the next stage of the subtree of the physical multicast tree, from that link to the leaves. At the termination of the **Tree_feasible**, each undominated set on the root access link has a replica of the physical multicast tree associated with it. Each undominated set on any other link is associated with a replica of the subtree of the physical tree. By following these associated trees, we can extract all undominated flows.

The procedure is given as input.

A description of the physical multicast tree with the bandwidth associated with each link

An $L$-vector, where $L$ is the number of streams offered by the root, and the $i$'th component is the bandwidth requirement of stream $i$.

```
Tree_feasible(root_link, vector)
    for all child_link(root_link)
    link_feasible(child_link, vector)
    for all undominated sets
    append all child_links(child_link)
    if child_link is not a leaf
    for all undominated sets
    tree_feasible(child_link, undominated_set).
```

### 4.3.3 A Procedure for Evaluating All Feasible Flows

A flow is associated with a gain that is "earned" whenever a stream is delivered on a leaf. The gain per stream per leaf equals the bid on that leaf for that stream; and the gain of a flow is the sum over all leaves of the bids for all streams that reach the respective leaves. The flow to select is, of course, the one that has the maximum gain. The recursive procedure **Value** tracks all flows from their undominated sets on the root link. It uses the fact that the value of a set on a link is determined by the values of all of its undominated subsets on the child links, according to the following rules:

1. On each child link, the undominated subset with the maximum gain is selected.
2. The sum of the values found in item 1 above over all child links is the value of the set on the parent tree.

To find the undominated flow with the maximum gain, we use the following rules:

1. The gain of an undominated set on a leaf is the sum of the bids for the elements of the set on that leaf.
2. The gain on a link is the maximum of the gains of all undominated sets on that link.
3. The gain of an undominated set on a nonleaf link is the sum of the gains on all the child links associated with that set.

In a semiformal manner, the procedure **Value** that computes this maximum gain is given in the following recursive form:

```
Value(root_link, set)
    If link == leaf
    $V = \sum_{stream \in set} b_s$
    Else
    $V = \sum_{\mathrm{down\_link}} \max(\mathrm{Value(down\_link, set)})$.
```

Here $b_s$ is the bid for stream $s$, and the summations in the first and second expressions are over all undominated subsets that reach the leaf, and over all children links in the extended tree, respectively.

### 4.3.4 An Example

The following example illustrates the operation of the algorithm. Figure 12 shows the physical multicast tree, including the streams (1,2,3,4) offered by the source and their respective bandwidth demands, namely [1,2,4,5]. The links' available capacities are enclosed in angular brackets, and the bids for each stream at each of the destinations A, B, C, and D, are enclosed in curly brackets. For example, there is sufficient path capacity to deliver stream 2 to destination A, and if that stream is delivered to that destination, the source's gain is 1.

**Link_feasible** operates as illustrated in Figure 13. It begins by examining row 4, with the full set of offered streams, i.e., (1,2,3,4). Since the total bandwidth demand of 12 exceeds the link capacity of 8, the procedure moves to examine row 3, which contains the sets of three streams. Of these, the sets (1,2,3) and (1,2,4) are feasible, hence undominated, while the other two are infeasible. When the procedure examines row 2 it examines the set (3,4), since the other sets of

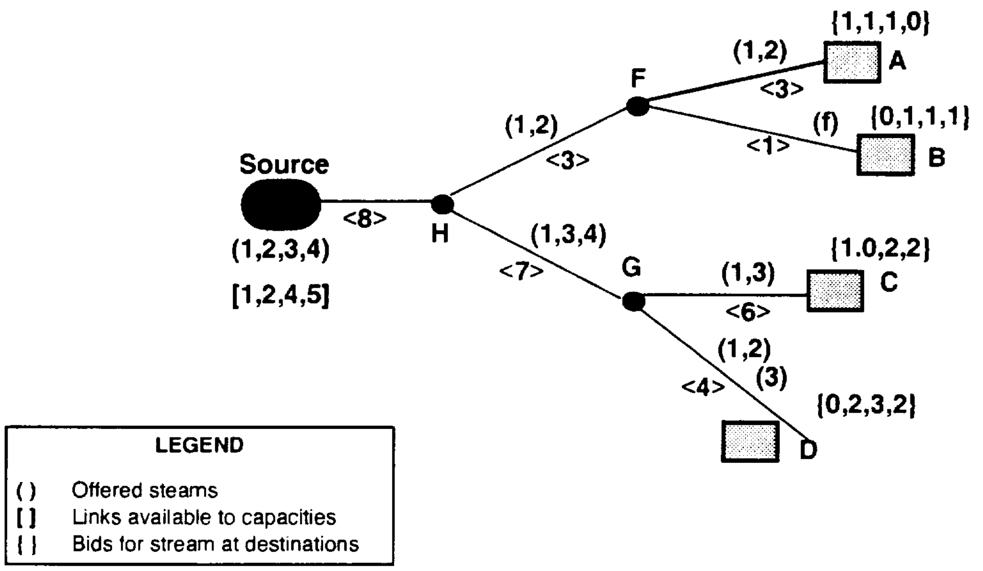


Figure 12. A Physical Multicast Tree

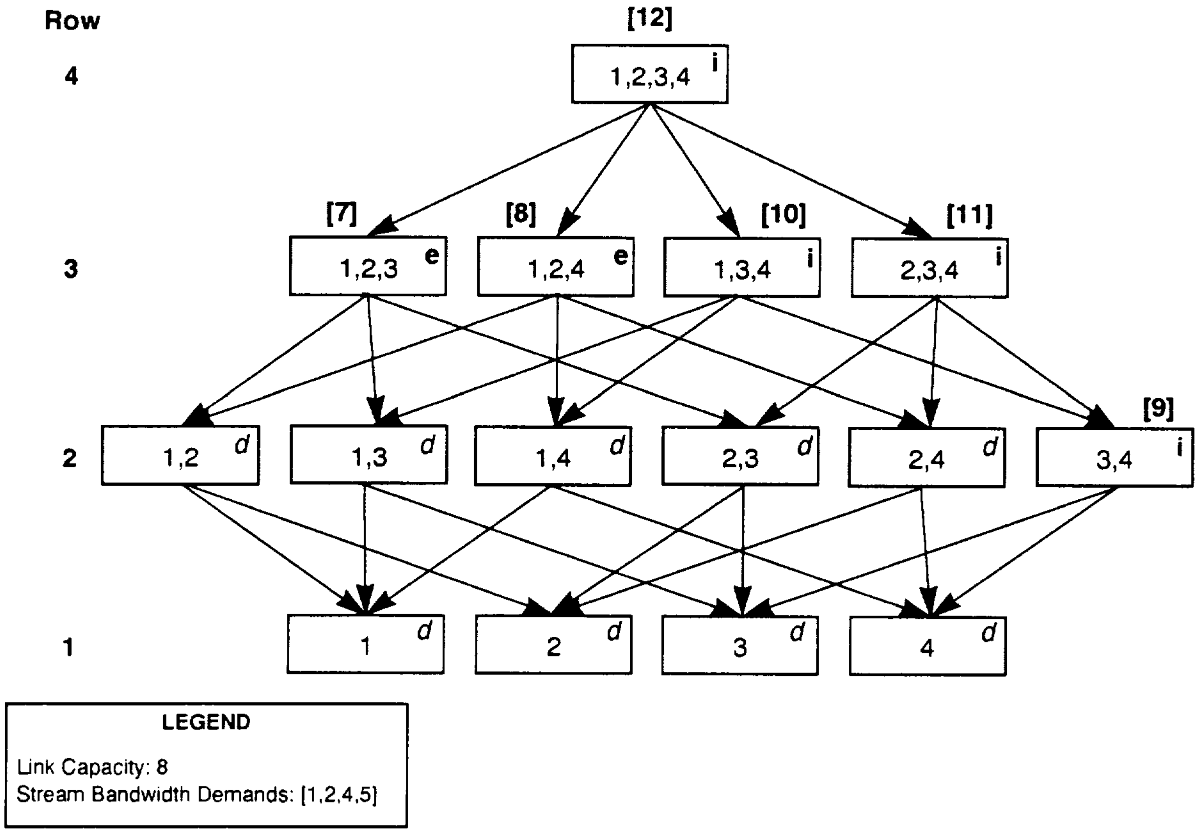


Figure 13. Finding Undominated Sets Over a Link

Hamming weight 2 are dominated by the undominated sets that have already been found. Since the set (3,4) is infeasible, and since all sets in the last row are dominated, the procedure terminates. The output of the procedure in this case consists of the two undominated sets found above.

**Tree_feasible** begins at the root and applies **Link_feasible** to the access link (leading to node H), resulting in two undominated sets, namely (1,2,3) and (1,2,4), with total bandwidth requirements of 7 and 8, respectively. For each of these sets, a replica of the two links HE and HG is appended to node H, as shown in Figure 14: i.e., the links HF1, HG1 and HF2, HG2 are associated with the sets (1,2,3) and (1,2,4), respectively. Using these sets, the procedure **Link_feasible** finds the undominated subsets to be (1,2) and (1,2,3) on links HF1 and HG1, respectively. Similarly, the set (1,2,4) is used on HF2, where the single undominated set (1,2) is found, and HG2, where there are two undominated sets, (1,4) and (2,4). Associated with each of these sets is a replica of the links one step down the tree from the link on which they are computed. Figure 15 shows the full extended tree, where there is one replica of the subtree at nodes F1, G1, and F2, because at each of those nodes a single undominated tree is offered to the subtree. At G2, on the other hand, two replicas of the down tree are appended, one each for the sets (1,4) and (2,4) that are being offered at G2. The bandwidth values of the down-tree links further reduce the number of streams that can be delivered. The sets marked on the leaves of the extended tree in Figure 15 constitute all the undominated sets that can be delivered.

The procedure **Value** translates the bids at the leaves to numerical values at all links. For example, since link G2D22 is a leaf, delivering stream 2 results in a gain of 2, which is the bid that destination D assigns to stream 2. On link G2C22, one of the two undominated sets (2) and (4) can be delivered, with resulting gains of 0 and 2, respectively, implying that it is preferable to deliver set (4). Thus, the total gain of the set (2,4) on the link HG2 is 4, while a similar argument shows that the corresponding gain of (1,4) on that link is 3. That is, the former set is preferable to the latter on link HG2. The total bids for sets (1,2,4) and (1,2,3) on the root access link can be calculated in a similar fashion to be 6 and 7, respectively, thereby making the latter set more attractive. The selected distribution is, therefore, (1,2) on link HF in the physical tree and (1,2,3) on HG. The remaining stream assignments are (1,2) on FA, (1) on FB, (1,3) on GC, and (1,2) on GD. A final correction is made by observing that stream (1) results in 0 gain on FB and GD and can therefore be eliminated from those links without penalty.

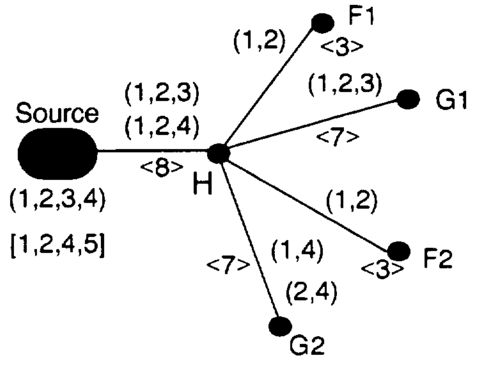


**Figure 14. Initial Buildup of the Extended Tree**

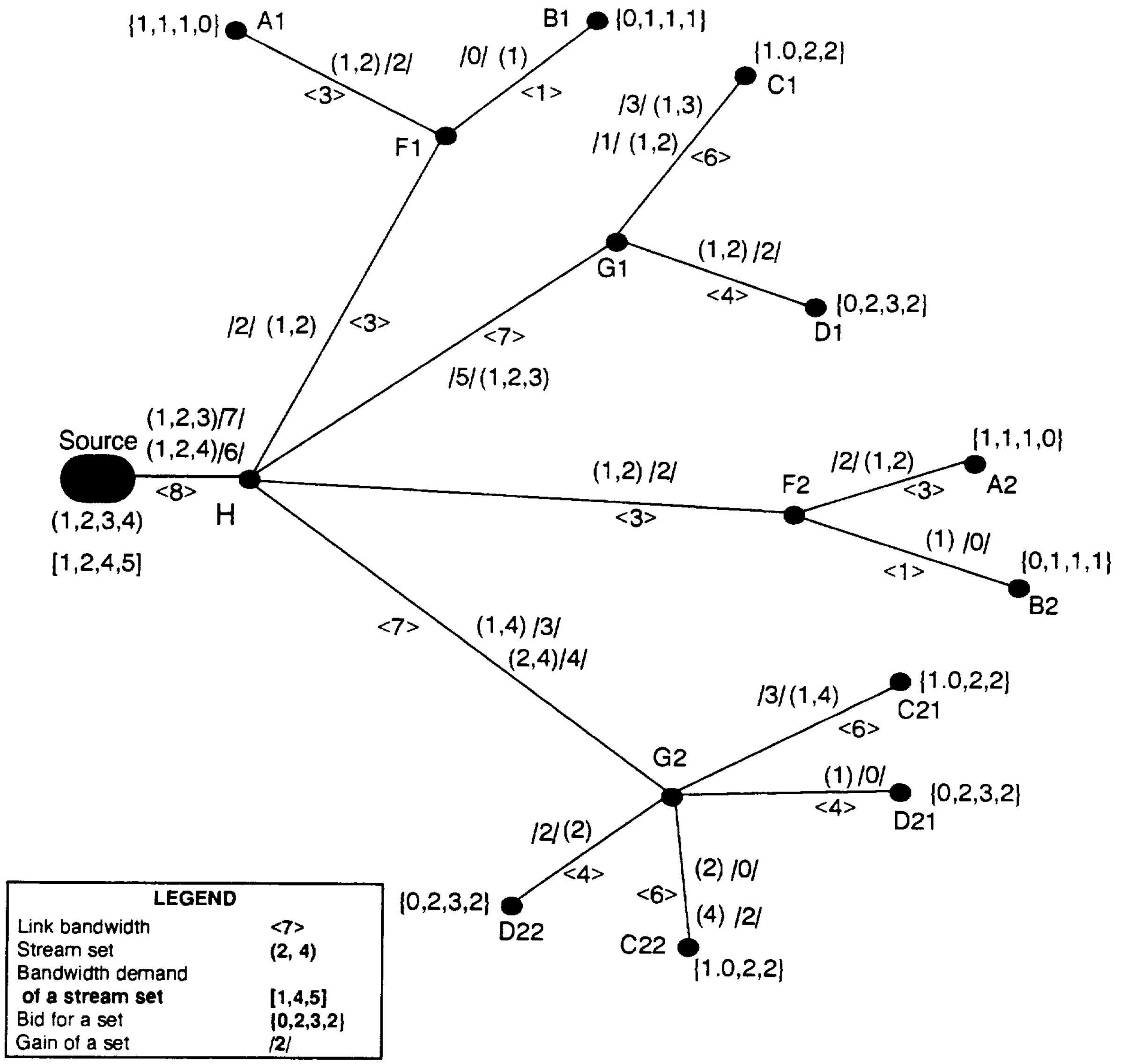


**Figure 15. The Extended Tree and Flow Gains**

## 4.4 CORRECTNESS PROOF

To prove the correctness of the algorithm, it must be shown that of all the flows generated by the algorithm, the one with the highest gain is selected; and that there is no other feasible flow with a higher gain. Our approach for the proof is, therefore, to show that the following hold true:

1. If a flow $A$ is not included in the algorithm's outcome, either that flow is infeasible, or a feasible flow $B$ that dominates $A$ is found by the algorithm. Notice that the latter implies that the gain of flow $B$ is equal to or larger than that of $A$.
2. Of all the flows in the algorithm's output, the one with the highest gain is selected.

A feasible flow **X**, as defined in Subsection 4.1.2, can be represented by a set of vectors **Xj**, one for each of the $L$ links of the multicast tree, $X_j = (X_{1,j}, \ldots, X_{N,j})$, with 0-1 values that satisfy constraints (21) and (22) for continuity and bandwidth feasibility, respectively. Flow **X** is dominated by flow **Y** if $X_{i,j} \le Y_{i,j}$ for all $i, j$.

A flow **Y** is included in the algorithm's output if

- Its vector $\mathbf{Y_s}$ that corresponds to the root access link is one of the sets listed by **Link_feasible** for that link
- Its vector $\mathbf{Y_j}$ that corresponds to a child of the root access link is one of the sets listed by **Link_feasible** on the extended tree, for that link on the subtree that is associated with the aforementioned set on the access link
- Its vector $\mathbf{Y_i}$ that corresponds to a link in the physical multicast tree is one of the sets listed by the algorithm for that link on the subtree associated with the set that corresponds to that flow on the parent of that link in the extended tree.

Let **X** be a feasible flow on the physical multicast tree, and consider its vector $\mathbf{X_s}$ for the root access link. Since the procedure **Link_feasible** lists on that link all undominated subsets of the $N$ streams offered by the source, it must select at least one such set $\mathbf{Y_s}$ such that $X_{i,s} \le Y_{i,s}$ for all $i = 1, \ldots, N$. If there are more dominating vectors, we arbitrarily choose one of them. Consider now a child link $j$ on the extended tree that is associated with the selected set $\mathbf{Y_s}$; and compare $\mathbf{X_j}$ to all subsets of $\mathbf{Y_s}$ on that link. If there is a subset $\mathbf{Y_{sl}}$ such that $X_{i,j} \le Y_{i,sl}$ for all $i$, we continue to check the other child links. If we can identify such dominating sets on all of the child links, we continue to the respective child links associated with the identified subsets. If in this process we reach all leaves, the collections of subsets **Y** so identified constitute a feasible flow that dominates flow **X**.

Consider now the case in which we follow the selected subsets as described above to reach link $j$, where no dominating subset can be found. Since a dominating set $\mathbf{Y_k}$ was found on the parent link $k$ (by the above process), and since the procedure **Link_feasible** lists all undominated subsets of $\mathbf{Y_k}$ on the child link $j$, the discrepancy means that $\mathbf{X_j}$ is not a subset of any of the sets listed on link $k$. But, since $\mathbf{X_k}$ is dominated by $\mathbf{Y_k}$, $\mathbf{X_j}$ cannot be dominated by $\mathbf{X_k}$, which contradicts the feasibility assumption for **X**.

Accordingly, we can restrict our attention to the flows listed on the extended tree. Suppose that a flow **Y** achieves a higher gain than the **Z** identified by the procedure **Value**. Consider the subtree of the extended tree with which flow **Y** is associated. To be higher than **Z**, the gain of $\mathbf{Y_s}$ must be larger than that listed for $\mathbf{Z_s}$. But the procedure **Value** computes a smaller gain for $\mathbf{Y_s}$ than for $\mathbf{Z_s}$; therefore $\mathbf{Y_s}$ must be smaller than the gain of **Y**. That is, on one of the child links of the subtree associated with $\mathbf{Y_s}$, a smaller than maximum value must have been selected. But this is impossible, since on each link the procedure **Value** selects the subset with the maximum value, where the value of each vector is computed as the sum of the values of the vectors on the associated children links. These arguments show that the algorithm does select the flow with the maximum gain, and is therefore correct.

## 4.5 COMPLEXITY ANALYSIS

*Lemma:* If an $N$-stream set is considered by the procedure **Link_feasible** over a link, the maximum number of undominated sets is $\binom{N}{N/2}$.

*Proof:* Suppose that all streams have the same bandwidth requirements $w$. If the link bandwidth is $W$, then all undominated sets contain $\lfloor W/w \rfloor$ streams, since all sets with that number of streams are feasible, no set with more streams is feasible, and all those sets with fewer streams are dominated. That is, the number of undominated sets in this equal-bandwidth case is $\binom{N}{\lfloor W/w \rfloor}$, and the maximum such number occurs when $\lfloor W/w \rfloor = N/2$.

Suppose now that one stream has a larger bandwidth requirement than the other: e.g., stream 1 has $k$ times the bandwidth requirement of each of the other streams. The maximum number of undominated sets in this case is $\binom{N-1}{\lfloor N/2 \rfloor} + \binom{N-1}{\lfloor N/2 \rfloor - k} \le \binom{N}{\lfloor N/2 \rfloor}$.

Thus, when the procedure **Link_feasible** is applied to a link on which $K$ streams are offered, in the worst case the procedure is required to check $2^K$ subsets and yields as its output $\binom{K}{K/2}$ sets, each of which is being applied to each child link.

Suppose the tree has $M$ stages and at every stage each link has $D$ child links. The source offers $N$ streams to the links on the first stage. **Link_feasible** is applied to each of the $D$ first-stage links and on each it checks $2^N$ sets and generates $\binom{N}{N/2}$ undominated sets. The second stage of the extended tree has $\binom{N}{N/2}D$ links, and on each link **Link_feasible** considers a set of size $N/2$: i.e., it checks $2^{N/2}$ sets, and generates $\binom{N/2}{N/4}$ sets to be tested on the third stage.

The above arguments lead to the following estimate of the computational complexity of **Tree_feasible**:

$$D(2^N + 2^{N/2}\binom{N}{N/2} + 2^{N/4}\binom{N}{N/2}\binom{N/2}{N/4} + \dots + 2^{N/2^{M-1}}\binom{N}{N/2}\binom{N/2}{N/4}\dots\binom{N/2^{M-2}}{N/2^{M-1}}) \quad (25)$$

This computation is less complex than checking all the sets on all links, with the resulting complexity $2^{NL}$, where $L$ is the number of links, given by

$$L = D + D^2 + D^3 + \dots + D^M = D\frac{1 - D^M}{1 - D}.$$

## 4.6 CONCLUSION

Protocols for exchanging real-time multimedia information in a session with multiple participants are likely to be essential to such important applications as teleconferencing and remote learning. Means for the efficient distribution of multimedia traffic are of particular importance for heterogeneous multicast sessions, which are conducted over networks with nonuniform link capacities and in which users are allowed to specify individual demands for the streams they wish to receive.

In this section we have addressed the problem of optimal selection and multicast distribution of multimedia streams in heterogeneous sessions. Since users compete for network resources, each user offers a certain bid for each stream it requests, and the source gains the respective bids whenever a stream is delivered to a user. Bids, which are nonnegative numbers, represent users' interest in the streams, and are used to measure the relative satisfaction of the users with the streams delivered to them. Thus, distribution algorithms strive for higher gains, and the optimal distribution is the one that results in the maximum gain.

We have assumed that all the streams are distributed over a single multicast tree. Such a tree may be, for example, the maximum bandwidth tree that results in paths with maximum capacity from the source to each destination. The algorithm is suitable for centralized implementation because it requires information about bandwidth availability on the tree as well as knowledge of users' demands and bids.

We have formulated the problem of multimedia distribution over a tree as 0-1 integer programming, and have developed and illustrated the operation of an algorithm for optimal distribution. The algorithm has much smaller computational complexity than the direct stream-subset enumeration and evaluation. Still, the resulting complexity is rather high. Further reductions in complexity can be expected from approximation and heuristic algorithms using the greedy approach. Several such algorithms, which we have developed, will be presented in a forthcoming paper.

Future work on this problem will include the development of fast, parallel algorithms for approximate solutions for networks with large values of $N$, $L$, and $M$; and incorporation of heterogeneous, bursty traffic sources into the model.

# 5 ALGORITHMS FOR INCREMENTAL CHANGES TO A MAXIMUM BANDWIDTH TREE FOR HETEROGENEOUS MULTICAST

Multicast protocols were traditionally developed under the assumption that all recipients receive all the information emitted by a source. To that end, a tree is established with the source at its root and reaching all destinations; every inner node on the tree replicates all traffic from the root to all outgoing branches. Shortest-path or minimum-cost trees are the most common approaches.

In many heterogeneous networks, users differ widely in the bandwidth available to them, and when the distributed traffic consists of real-time streams, it is likely that not all intended participants have the bandwidth and processing capability for receiving the full streams. Insisting on uniformity in multicast distribution in such cases raises a serious dilemma: either the more limited users, which cannot receive the full stream, are excluded; or information must be overly compressed, thereby penalizing the more capable users.

An approach termed heterogeneous multicast (HMC) that circumvents the above dilemma was proposed by Shacham [1992]. In this approach, based on layer encoding of real-time streams, the quality of a received stream is a function of the number of layers delivered. The stream distribution is done over a maximum-bandwidth tree (MBT), consisting of the path of maximum capacity available for each destination, on which the maximum number of layers are delivered to their respective destinations. Notice that although the reception quality varies from destination to destination, each gets the highest quality possible.

Multicast routing can be computed for a complete set of destinations or incrementally. In the first case, which is considered in Shacham's 1992 paper, an MBT is constructed for a given source, a layered stream with its bandwidth characteristics, a network topology, a set of destinations, and their bandwidth requirements. This type of calculation is suitable for the beginning of a multicast session in which all preregistered participants are to be connected. This approach for calculating a multicast distribution tree to a set of destinations has received the main attention in multicast routing research. Most of the approaches proposed thus far focus on minimizing the cost to the network or the users, which has led to the computation of Steiner or to minimum delay trees [Ghanbari 1989].

Real-time multicast sessions, such as teleconferencing, are expected to be dynamic, with the number of destinations, their bandwidth requirements, and the bandwidths available varying with time. In such an environment the set of paths to the destinations is likely to change in response to destinations joining the session, leaving it, or changing the number of layers they request during the session. Instead of computing a new MBT each time the dynamics change, which may result in many changes to the existing MBT, it is desirable to make incremental changes to the existing MBT, to accommodate the current dynamics and at the same time minimize the changes to the MBT. These incremental changes to the HMC distribution paths are the subject of this paper.

In general, updating the HMC distribution paths requires two steps:

1. Computation of a path (or paths)
2. Allocation of resources, e.g., bandwidth reservations, along the path.

The focus of this report is on the first part, that is, algorithms for computing routes that satisfy users demands. Most of the work on multicasting assumes a static environment [Bharath-Kumar and Jaffe 1993; Doar and Leslie 1993; Waxman 1988; Waxman 1993]. Karlsson and Vetterli [1989] and Ghanbari [1989] consider a dynamic environment in which hosts are added and removed from a nonheterogeneous multicast connection. The routing algorithm attempts to minimize network cost. The resultant dynamic Steiner tree problem is solved by means of heuristics, and the performance of these heuristics is illustrated. Our problem is different from the above, in that we consider heterogeneous multicast. In addition, our objective is different. We are interested in determining the maximum-bandwidth shortest paths to each destination. Papadimitriu and Steiglitz [1982] present an ad hoc resource reservation protocol that supports multicast applications. This work assumes the existence of an underlying routing strategy and is as such not related to our work.

The emphasis in this report is on extending the MBT to include a new destination. Other dynamics are simpler to handle and require changing only the forwarding tables at the nodes. Extending an MBT to include a new destination must be done in two phases:

1. Obtain a path to the new destination
2. Restore the tree structure to the routing paths, in case the first phase destroys the tree structure of the existing MBT, since it is desirable that at the conclusion of phase 1, the collection of paths is still a tree.

First, we present centralized algorithms that use the link-state approach to route computation by assuming that the full network topology and bandwidth availability are known to the processing units that compute the MBT. These processing units also populate forwarding tables at each node in the MBT with information about the number of signal layers to be sent to each downstream node. Also, the routing tree obtained is a source-oriented tree, even if the algorithm computations are started from a destination node.

Algorithms for the first phase are Dijkstra-like [Jayant 1992]; in which both the cost function and the labelling rule change to suit our objective. Two types of algorithms are presented, in which

1. The maximum-bandwidth path to the new destination is obtained subject to the condition that no changes are made to the existing MBT
2. The maximum-bandwidth path to the new destination is obtained with minimal changes to the existing MBT.

For each of these two types we present two algorithms, in which

1. The algorithm computation is started from the source node
2. The algorithm computation is started from the new destination: the advantage of this approach is that on average it is likely to have lower computational complexity.

Next, we present distributed asynchronous algorithms to construct an MBT and to make incremental changes to it. The advantage of using such algorithms is, of course, that very little information needs to be stored at the network nodes. Also, such algorithms respond more quickly to network dynamics. Information need be exchanged only between adjacent nodes.

The algorithms presented are Bellman-Ford-like in that both the cost function and the labelling rule change to suit our objective. In addition to a distributed asynchronous algorithm to construct an MBT at the start of a multicast session, two distributed asynchronous algorithms are presented to add a new destination to the multicast session. The first algorithm obtains the maximum bandwidth path to the new destination, subject to the condition that no changes are made to the existing MBT; the second obtains the maximum-bandwidth path to the new destination with minimal changes to the existing MBT.

The rest of this section is organized as follows. In Subsection 5.1, we present the network model. In Subsection 5.2, we review the centralized algorithm [Shacham 1992] that is used to obtain an MBT for heterogeneous multicast in a static environment, since this algorithm forms the basis of other centralized algorithms for making incremental changes to the MBT. These algorithms are described in Subsection 5.3. In Subsection 5.4, we present a distributed asynchronous algorithm to construct a MBT, given a source, a layered stream with its bandwidth characteristics, the network topology, a set of destinations, and their bandwidth requests. In Subsection 5.5, we present distributed asynchronous algorithms to make incremental changes to a MBT. Finally, in Subsection 5.6, we present a summary of this section.

## 5.1 NETWORK MODEL

The network is modelled by a graph $G = (V, E)$, where $V$ and $E$ are sets of nodes and links, respectively. One of these nodes is the source node and $D \subset V$ is the set of multicast destinations for a multicast session. We focus on one hierarchical multicast session and characterize each link $(i, j) \in E$ by the capacity, $b_{ij}$, available on this link for the session. To extend the model to incorporate the maximum bandwidth, say $W_s$, at which the source delivers the signal, a dummy node replaces the source node in the graph and the source node is connected to this dummy node by a link of capacity $W_s$. In addition, to incorporate the destination's bandwidth requirements, the graph $G$ is augmented by $M$ nodes and $M$ links, where $M$ is the number of multicast destinations. For each destination $d$, $d \in D$, a node $d'$ replaces $d$ in the graph and $d$ is connected to $d'$ by link $(d, d')$ of capacity $W_d$, $W_d$ being the bandwidth at which destination $d$ desires the multicast signal.

The source generates its signal in a hierarchy of layers, say $K$, where layer 1 (the lowest layer) represents the basic signal and layer $i$, $1 < i \le k$, improves upon the quality of the signal constructed from layers $1, 2, \ldots, i-1$. We assume that the maximum number of layers, $K$, and the total bandwidth of the full signal are both fixed, but that the bandwidth assigned to each layer can be modified by the source. The bandwidth assigned to each layer can be set to maximize the quality seen by the set of destinations. Details of this optimization procedure are provided by Shacham [1992]. These signal layers are then delivered over an MBT consisting of the path of maximum capacity available for each destination, on which the maximum number of layers are delivered to the respective destinations. Note that although the reception quality varies from destination to destination, each gets the highest quality possible.

## 5.2 ROUTE FINDING ALGORITHMS FOR HETEROGENEOUS MULTICAST IN A STATIC ENVIRONMENT

Given the aforementioned model, we use a Dijkstra-like algorithm to compute the maximum single-path bandwidth to all destinations. Each node $i$ is assigned three labels, $p_i$, $B_i$ and $H_i$, where $p_i$ is the current preferred neighbor at $i$, $B_i$ is the maximum path capacity from the source to the node $i$ along the path through $p_i$, and $H_i$ is the hop count (number of links) from the source to $i$ along this path. The labels are updated as the algorithm progresses. At each step of the algorithm, one node becomes permanently labeled. Once a node's label is permanent, the node is included in a set $P$. Assume that the source node is 1.

**Initialization Step**. $P = \{1\}$, $B_1 = \infty$, $H_1 = 0$, $p_1$ is left undefined, since we terminate a path at the source and hence there is no preferred neighbor at 1. $B_j = b_{1j}$, $H_j = 1$, and $p_j = 1$ for $j \neq 1$.

**Step 1**. (Permanently labelling a node) Find $i \notin P$ such that

$$B_i = max_{j \notin P} B_j$$

If more than one node satisfies the above equation, choose the node with the smallest hop count to the source for permanent labelling and inclusion in P. Node $i$ is now permanently labelled and included in set $P$. Set $P = P \cup \{i\}$. If $P$ contains all the destination nodes, then stop.

**Step 2**. (Updating of labels) For all $j \notin P$ set

$$B_j = max\{B_j, min\{B_i, b_{ij}\}\}$$

If $B_j$ is the larger of the two terms in the above equation, the other labels $H_j$, $p_j$ at node $j$ are left unchanged; otherwise, they are changed as follows:

$$H_j = H_i + 1$$

$$p_j = i$$

Go to Step 1.

The following can be proved for the above algorithm, as in Jayant [1992]:

- $B_i$ label at each node, $i$, indicates the maximum bandwidth of all paths from the source to that node.
- The permanent label values are assigned to nodes in a nonincreasing sequence of their permanent $B_j$ values.
- The maximum-bandwidth path from the source to a destination is obtained by starting at the destination and connecting nodes to their preferred neighbors until the source node is reached.
- If there is more than one maximum-bandwidth path from the source to a destination, the one with the smallest hop count is chosen.
- The set of maximum bandwidth paths from the source to all the multicast destinations forms a tree, termed a maximum bandwidth spanning tree (MBT).

Using $B_i$ values to determine the number of signal layers to be transmitted on a link may create a situation where more layers are transmitted on a link than are necessary. The above Dijkstra-like algorithm is followed by an $O(N)$ bandwidth-trimming procedure. In this procedure, starting from each destination, bandwidth labels are propagated to the upstream nodes towards the source. At each node the bandwidth label is set to be equal to the maximum-bandwidth label of each downstream node. The details of this procedure are provided by Shacham [1992].

The worst-case computation complexity of the Dijkstra-like algorithm is $O(N^2)$. The bandwidth trimming procedure takes $O(N)$ steps in the worst case, since at most N nodes are visited. Thus, the overall worst-case computation complexity for the route-finding algorithm is $O(N^2)$.

## 5.3 ALGORITHMS FOR INCREMENTALLY CHANGING THE MBT TO ACCOMODATE A DYNAMIC ENVIRONMENT

At the beginning of a multicast session, an MBT can be constructed via the algorithm described in the previous section, for a source, a layered stream with its bandwidth characteristics, a network topology, a set of destinations, and their bandwidth requirements.

As we mentioned earlier, a multicast environment will most likely be dynamic where the number of destinations involved in the multicast session, the number of layers requested by a destination, and the bandwidth available for this multicast session on each link in the network may all vary with time. Changes to a current multicast session are requested by a destination. The following requests can be made:

1. A destination may request to be deleted from the multicast session.
2. A destination may request that the number of signal layers delivered to it be reduced.
3. A destination may request that the number of signal layers delivered to it be increased.
4. A new destination may request to be added to the multicast session.

A possible way to effect these changes is to compute a new MBT each time a request for a change arrives, and switch routing paths from the old MBT to the new MBT. The problem with this approach is that it may cause disruptions to nodes on the old MBT. A disruption at a node occurs when paths from the source to the node along the old MBT and the new MBT join the node via different links. A disruption at a node is undesirable because it involves choosing one of the two incoming links on which to receive the signal, which may cause out-of-order packets, switching delays, and the like at this node, and these undesirable effects may be propagated to the destination node, downstream from this node. One of the reasons why paths from the source to a node could be different along the old and new MBTs is that the bandwidth available for this multicast session on various links in the network may have changed between the time the old MBT was computed to the time the new MBT is computed.

A more interesting and desirable way to effect the changes is to do so with no disruptions or with minimal disruptions to nodes on the old MBT. Thus only incremental change occurs to the old MBT, each time a request is served. In what follows we shall describe algorithms that effect these changes with no disruptions or with minimal disruptions to nodes on the old MBT.

While making incremental changes to the MBT, we follow the link-state approach by assuming that the full network topology and bandwidth availability are known to the processing units that make the changes to the MBT. With each incremental change, these processing units also populate forwarding tables at the appropriate nodes in the MBT with information about the number of signal layers are to be sent to each downstream node. A we mentioned earlier, the MBT is a source-oriented tree, even if the algorithm's computations start from a destination node.

Request (1) can be considered to be a special case of request (2), since deleting a destination from a multicast session amounts to reducing the number of signal layers to this destination to zero. To accommodate requests (1) and (2), a message from the destination is sent upstream, towards the source to each node along the multicast tree, requesting a reduction in the number of signal layers that are forwarded to this destination. This request propagates up the multicast tree towards the source until it reaches a node that forwards more signal layers than are desired by the requesting destination, along another tree branch. This algorithm has a worst-case computation complexity of $O(N)$, $N$ being the number of nodes in the multicast tree, since at most $N$ nodes are visited.

In the case of request (3), in the simplest case, there is sufficient residual capacity on the multicast tree links to support the new demand. A message from the destination is sent upstream to each node along the multicast tree towards the source, requesting an increase in the number of signal layers that are forwarded to this destination. This request propagates up the multicast tree towards the source until it reaches a node that has the desired number of signal layers. This algorithm also has a worst-case computation complexity of $O(N)$, $N$ being the number of nodes in the multicast tree, since at most $N$ nodes are visited.

To effect the above changes, no change in the old MBT is required. These algorithms merely update the forwarding tables at the MBT nodes.

The task of meeting requests 1, 2, and 3 is more challenging when the available bandwidth on the tree is insufficient for delivery of the requested number of signal layers to the requesting destination. In this case, for purposes of route computation, this destination node is deleted from the old MBT and treated as in the case of request 4, where a new destination requests to be added to the multicast session.

To accommodate request 4, we present two types of algorithms.

- The first type follows a "no disruption" approach: that is, it does not change the structure of the existing tree. These algorithms find a path to the new destination from a node of the existing tree such that no other tree node is on the path. Off all such paths, the algorithm finds one that can deliver to the destination the largest number of stream layers.
- The second type of algorithms finds a path that can deliver the maximum number of layers to the requestor even if that path passes through several tree nodes, thereby creating multiple intersecting paths and destroying the tree structure. These algorithms destroy the tree structure at a minimal number of nodes, thus causing a minimal number of disruptions to the current multicast destinations. A procedure to restore the tree structure to the routing path follows the conclusion of the above algorithms.

We have developed two algorithms for each of these two types.

- First, where the algorithm computation is started from the source node
- Second, where the algorithm computation is started from the new destination; the advantage of the latter is that on average it is likely to have lower computational complexity.

All the algorithms are modifications of the Dijkstra-like algorithm described above in Subsection 5.2, in which both the cost function and the labelling rule change to suit our objective. In cases where disruptions occurs to the old MBT, an algorithm to restore the tree structure to the routing paths follows the Dijkstra-like algorithm.

In the following subsections, we will describe these algorithms in detail.

### 5.3.1 Source Initiated Without Disrupting Old MBT

The Dijkstra-like algorithm is run on a network graph modified by deleting all those incoming links at an old MBT node that are not on the old MBT. Deleting all such links ensures that the old MBT is not disrupted. The algorithm is followed by the bandwidth trimming procedure, as in Subsection 5.2.

The worst-case complexity of the algorithm is $O(N^2)$: the modified graph is obtained in at most $O(N^2)$ steps, since it involves examining at most all the edges of the network graph, and the Dijkstra-like algorithm takes at most $O(N^2)$ steps.

### 5.3.2 Destination Initiated Without Disrupting Old MBT

A Reverse-Dijkstra-like algorithm is used on the network graph to find the maximum-bandwidth path from the destination to the old MBT nodes. Permanent labels are assigned to nodes in a decreasing order of bandwidth to the destination. The algorithm terminates when it first permanently labels an old MBT node. When labeling a node that does not belong to the old MBT, the algorithm considers the full bandwidth available from the node to the destination; however, when labelling a node on the old MBT, the algorithm computes the minimum of the following:

- The bandwidth of the path from the node to the destination.
- The bandwidth of the old MBT path from the source to the old MBT node, which is the sum of the bandwidth currently used to deliver stream layers and the available capacity on the tree path from the source. We assume that the $B_i$ label on all old MBT nodes already has this information.* Let us denote the $B_i$ label on old MBT nodes $oldB_i$, to distinguish it from the $B_i$ label that is assigned to MBT nodes by the reverse-Dijkstra-like algorithm.

The permanent bandwidth labels on old MBT nodes are retained because they are used to forward signal layers to the old MBT nodes.

*This label updating could be accomplished, for instance, by a modified Dijkstra-like algorithm run on the modified network graph, as described in Subsection 5.3.1; the algorithm is triggered when a session using one or more MBT links concludes.

Let us denote the destination node as node 1.

**Initialization Step**. $P = \{1\}$, $B_1 = \infty$, $H_1 = 0$, $p_1$ is left undefined, since we terminate a path at the destination and hence there is no preferred neighbor at 1. For node $j$, $j \neq 1$, if node $j$, $j \neq 1$, then $H_j = 1$, and $p_j = 1$. Also, if node $j$, $j \neq 1$ is not a node on the old MBT, $B_j = b_{j1}$. If node $j$, $j \neq 1$ is a node on the old MBT, then

$$B_j = min'''' \{oldB_j, b_{j1}\}$$

**Step 1**. (Permanently labeling a node) Find $i \notin P$ such that

$$B_i = max_{j \notin P} B_j$$

If more than one node satisfies the above equation, choose the node with a smaller hop count to the source for permanent labelling and inclusion in P. Node $i$ is now permanently labelled and included in set $P$. Set $P = P \cup \{i\}$. Stop when any old MBT node is included in $P$.

**Step 2**. (Updating of labels) For all $j \notin P$ set

$$B_j = max\{B_j, min\{B_i, b_{ji}\}\} \qquad \text{if } j \text{ does not belong to old MBST}$$

$$= min\{oldB_j, max\{B_j, min\{B_i, b_{ji}\}\}\} \qquad \text{otherwise}$$

If $B_j$ is the larger of the two terms inside the *max* operator in the above equation, the other labels $H_j$ and $p_j$ at node $j$ are left unchanged; else they are changed as follows:

$$H_j = H_i + 1$$
$$p_j = i$$

Go to Step 1.

The maximum-bandwidth path from the new destination to the old MBT node is obtained by starting at the destination and proceeding through the connected nodes to their preferred neighbors until the first old MBT node is reached.

The path obtained to the destination as a consequence of the above algorithm, by extending the old MBT, is the maximum-bandwidth path possible without disruption of the old MBT. This is so because the permanent $B_i$ value at an old MBT node, obtained via the above algorithm, indicates the maximum-bandwidth path available from the source to the new destination along that node. In a Dijkstra-like algorithm, permanent node labels are assigned in a nonincreasing order of $B_i$'s; therefore, the first old MBT node that is permanently labelled is the one through which the maximum number of layers can be delivered from the source to the new destination.

A little reflection reveals that since we initiate the algorithm from the destination, we do not need any bandwidth trimming along the new path from the old MBT to the new destination. The permanent bandwidth label, obtained via the above algorithm at each node from the new destination along the maximum-bandwidth path to the node at which this path joins the old MBT, can be used to determine the number of signal layers to be transmitted on each link on the new path.

The worst-case computation complexity of this algorithm is again $O(N^2)$, which is the worst-case computation complexity of the reverse-Dijkstra-like algorithm.

### 5.3.3 Source Initiated with Minimal Disruption of Old MBT

Our objective here is to obtain a maximum-bandwidth path to the new destination, with a minimal number of disruptions to the nodes in the old MBT. We run the route-finding algorithm in two phases.

In the first phase, we obtain maximum-bandwidth paths to all the multicast destinations with minimal disruption to existing nodes in the old MBT. This procedure superimposes a new MBT over the old one, and may destroy the tree structure at certain nodes.

In the second phase, an algorithm is run to restore the tree structure to the routing paths.

We now describe the two phases.

**First Phase.** The old MBT node labels are retained, and a slightly modified Dijkstra-like algorithm is run to obtain a new MBT containing all the multicast destinations. The slight modification lies in a provision whereby if a choice exists between two maximum-bandwidth paths from the source to a destination with the same hop count, the path that causes the fewest disruptions of the old MBT nodes is chosen. This is accomplished by adding another label, $D_i$, to the set of labels at each node $i$. $D_i$ is the number of disruptions caused to the nodes of the old MBT along the path from the source to $i$. During the initialization step, $D_i$ is set to 0 for all nodes.

During step 1 of the Dijkstra-like algorithm, if more than one node satisfies the *max* operator, the node with the smallest hop count to the source is chosen for permanent labelling and inclusion in $P$. If more than one node satisfies the *max* operator and if more than one of these have the same hop count to the source, the one with the smallest value of $D_i$ is chosen. The algorithm terminates when the new destination is permanently labelled and included in the set $P$.

During step 2 of the Dijkstra-like algorithm, If $B_j$ is the smaller of the two terms in the equation, the labels $H_j$, $p_j$ at node $j$ are changed as before. In addition, if node $j$ is an old MBT node and if connecting it to node $i$ causes a disruption at node $j$, $D_j$ is set equal to $D_i + 1$. If node $j$ is not an old MBT node, $D_j$ is set equal to $D_i$. If $B_j$ is the larger of the two terms in the equation, the labels $H_j$, $p_j$, and $D_j$ at node $j$ are left unchanged. At the conclusion of the above algorithm, each node in the old MBT has two sets of labels: one set corresponds to the old MBT and the other new set corresponds to the maximum bandwidth path to the new destination. On the other hand, each new node added to the routing paths has just one set corresponding to the maximum-bandwidth path to the new destination.

**Second Phase.** During the second phase, an algorithm is run to restore tree structure to the routing paths, with minimal disruptions to the old MBT. This algorithm is implemented by modifying the bandwidth trimming procedure, as described by Shacham [1992]. Each destination, $i$, sends its $B_i$ label value upstream along the MBT.* At the upstream node $j$, after the node labels of all its downstream nodes are received, the $B_j$ label is set equal to the maximum of the bandwidth labels of each of its downstream nodes. If at node $j$ two incoming streams arrive along different links and if the stream along the old MBT has enough signal layers to satisfy the $B_j$ demand, a

*The new destination sends its only bandwidth label, i.e., the new bandwidth label; while the other destinations send their old bandwidth labels.

delete message is sent along the other path; otherwise, a delete request is sent along the old MBT path. This procedure restores the tree structure to the routing paths. Minimal disruptions are caused to the nodes of the old MBT, since the old MBT path is deleted only if it does not deliver the number of signal layers necessary to provide maximum bandwidth to the new destination. As in the bandwidth trimming procedure, $B_j$ value is propagated to the upstream node along the path that is retained. This step assures bandwidth trimming, if at all possible. The algorithm terminates when the source node examines the bandwidth labels received from each of its downstream nodes to see if it needs to reduce its bandwidth label value.

The phase-1 Dijkstra-like algorithm has a worst-case computation complexity of $O(N^2)$, and the phase-2 algorithm has a worst-case computation complexity of $O(N)$, since at most $N$ nodes are visited. Thus, the worst-case computation complexity of the above two-phase algorithm is again $O(N^2)$.

### 5.3.4 Destination Initiated with Minimal Disruptions to Old MBT

This algorithm is similar to the previous algorithm, the difference being that during phase 1, a reverse Dijkstra-like algorithm is run from the destination to the source. The old MBT labels are retained and the reverse-Dijkstra-like algorithm determines the maximum bandwidth path from the source to the new destination. The algorithm terminates when the source node is permanently labeled and included in the set *P*. Again, if there is a choice between two paths from the source to the destination, with the same bandwidth and the same hop count, the one with the fewest disruptions to the old MBT nodes is chosen. As before, the old MBT node labels are retained.

Let us denote the destination node as node 1.

**Initialization Step**. $P = \{1\}$, $B_1 = \infty$, $H_1 = 0$, $p_1$ is left undefined, since we terminate a path at the destination; hence there is no preferred neighbor at 1. For nodes, $j$, $j \neq 1$, $B_j = b_{1j}$, $H_j = 1$, and $p_j = 1$. For all nodes $i$, $D_i = 0$.

**Step 1**. (Permanently labeling a node) Find $i \notin P$ such that

$$B_i = max_{j \notin P} B_j$$

If more than one node satisfies the above equation, choose the node with smaller hop count to the source for permanent labelling and inclusion in *P*. If more than one node satisfies the above equation and more than one out of these nodes have the same hop count to the source, choose the one with a smaller value of $D_i$. Node $i$ is now permanently labelled and included in set *P*. Set $P = P \cup \{i\}$. Stop when the source node is included in *P*.

**Step 2**. (Updating of labels) For all $j \notin P$ set

$$B_j = max\{B_j, min\{B_i, b_{ji}\}\}$$

If $B_j$ is the larger of the two terms in the above equation, the other labels $(H_j, p_j, D_j)$ at node $j$ are left unchanged, else they are changed as follows:

$$H_j = H_i + 1$$

$$p_j = i$$

$$D_j = D_i + 1 \qquad \text{if link } (j,i) \text{ causes a disruption at } i$$

$$D_j = D_i \qquad \text{otherwise}$$

Go to Step 1.

During phase 2 of the algorithm, the maximum-bandwidth path from the new destination is followed to the source. Each time this path traverses an old MBT node, the old MBT node is examined to see if it receives the multicast signal via two different links. If this is the case, and if the bandwidth demand of the new path cannot be met along the old MBT path, a delete message is sent upstream along the old MBT path. If the bandwidth demand of the new path can be met along the old MBT path, a delete message is sent upstream along the new path.

The worst-case computation complexity of this algorithm is $O(N^2)$ as well, since the worst-case complexity of the phase-1 algorithm is $O(N^2)$ and the worst-case computation complexity of the phase-2 algorithm is $O(N)$.

## 5.4 DISTRIBUTED ROUTE-FINDING ALGORITHM FOR HETEROGENEOUS MULTICAST IN A STATIC ENVIRONMENT

Given the network model in Subsection 5.1, we use a distributed asynchronous Bellman-Ford-like algorithm to compute the maximum single-path bandwidth from all nodes to the source node. A straightforward adaptation of the Bellman-Ford algorithm would have us execute the following dynamic program algorithm at each node $i$. Without loss of generality, 1 is taken to be the source node.

$$\begin{aligned} B_i &= max_{j \in N(i)} \{min\{B_j, b_{ji}\}\} \ \ldots (i \neq 1) \\ B_1 &= \infty \end{aligned} \tag{26}$$

where $N(i)$ indicates the set of neighbors of $i$, and $B_i$ (the bandwidth label at node $i$) indicates the bandwidth of the maximum-bandwidth path from the source to $i$. Each node, from time to time, communicates its bandwidth label to all of its neighbors. The value of $B_i$ for the source node is kept fixed at infinity, while the initial values of $B_i$ at all other nodes can be any non-negative numbers. It is well known that the Bellman-Ford algorithm suffers from the so-called "bad news phenomenon" whereby the algorithm reacts (converges) slowly to a sudden increase in one or more link lengths. Unfortunately, the above adaptation of the Bellman-Ford algorithm suffers from a more severe bad-news phenomenon whereby the algorithm does not converge to the correct bandwidth labels if a sudden decrease in one or more link bandwidths occurs because of the presence of "route cycles." We illustrate this phenomenon with an example. In Figure 16, a part of a network is shown. The numbers on the arcs indicate available link bandwidths. Let us concentrate on the bandwidth labels for nodes $x$ and $y$. For link x-z, the available link bandwidth changes abruptly from 4 to 1. Before this change occurs, the above distributed algorithm converges to the correct bandwidth label values, i.e., $B_x = 4$; $B_y = 3$. After the change, the label values converge to $B_x = B_y = 3$ which are obviously incorrect values. The correct values are $B_x = B_y = 1$. This happens as a consequence of the "route cycle" x-y-x. We solve this problem as follows. Each node $i$, apart from the bandwidth label $B_i$, maintains a hop-count (to the source node) label, $H_i$, as well. $H_i$ for the source node is kept fixed at 0, while the initial $H_i$ values at all other nodes can be any non-negative numbers. The following dynamic program equations are executed at each node $i$.

$$p_i = j \in N(i) \text{ that""minimizes""} \alpha[1 + H_j] + max\left[\frac{1}{B_j}, \frac{1}{b_{ji}}\right] \ldots (i \neq 1) \tag{27}$$

$$\begin{aligned} B_i &= min\{B_{p_i}, b_{p_i i}\} \ldots (i \neq 1) \\ B_1 &= \infty \end{aligned} \tag{28}$$

$$\begin{aligned} H_i &= 1 + H_{p_i} \ldots (i \neq 1) \\ H_1 &= 0 \end{aligned} \tag{29}$$

where $\alpha$ is a positive constant. The above dynamic program equations find a path $P$ from the source node 1 to each node $i$, obtained by connecting preferred neighbors ($p_i$ at node $i$) from $i$ all the way to the source node, such that the cost function $\alpha H(P) + max_{l \in P} \frac{1}{b_l}$, $H(P)$ being the number of hops on path $P$ and $b_l$ being the bandwidth of link $l$ on path $P$, is minimized. Including the hop-count label in the dynamic program equations at each node ensures recovery from the bad-news phenomenon. The hop-count label of the nodes constituting a route cycle keeps increasing, and so does the cost of using them as a preferred neighbor. After some iterations, a "non-route-cycle" node

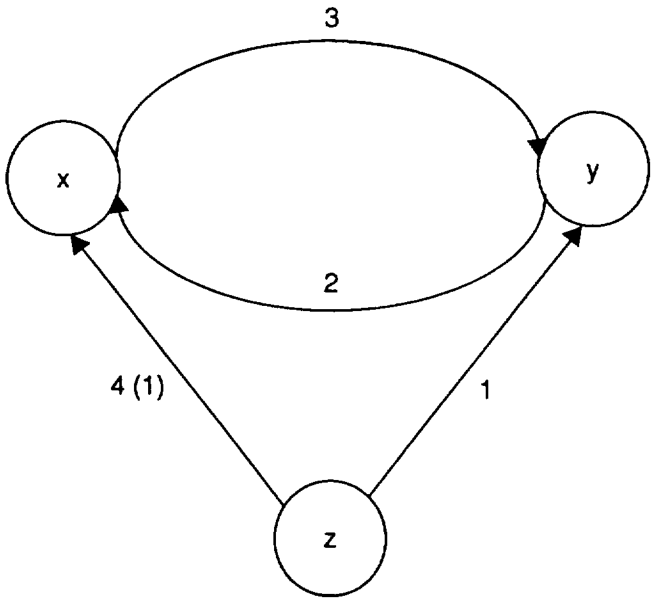


**Figure 16. "Bad News Phenomenon" Due to Sudden Decrease in Available Link Bandwidth**

becomes preferred at some node(s) constituting the route cycle; thus, the route cycle is broken. Similarly, in the example in Figure 16, if a small value is used for $\alpha$, the algorithm converges after some iterations to the correct bandwidth labels $B_x = B_y = 1$.

The value of the parameter $\alpha$ represents a tradeoff between the optimality (with respect to maximum-bandwidth paths) of the above algorithm and the time it takes for the algorithm to recover from the bad news phenomenon. A large value for $\alpha$ will enable quicker recovery from the bad news phenomenon, but an optimal solution (with respect to maximum-bandwidth paths) cannot be guaranteed. A very small value for $\alpha$ will result in an optimal solution for most practical cases but will imply that it may take many iterations of the above equations to recover from the bad news phenomenon. In addition, using the hop-count label in the above fashion has the desirable effect that the algorithm favors the path with fewer hops, if there are two paths of equal bandwidth to the source node.

Obviously, in order for the above scheme to be workable, the available bandwidth on links should not change too frequently relative to the speed of convergence of the above algorithm. Fortunately, this is often true in a practical network. The changes in available link bandwidth occur as a consequence of sessions being admitted to or dropped from the network. The time between such changes is, for the most part, greater than the time it takes for the algorithm to converge. Finally, the initial conditions of the algorithm require only that the label values at all nodes* be non-negative, so that when link bandwidths do change, the label values available at all the nodes are non-negative. Thus, it is not necessary to reinitialize the algorithm after each change in bandwidth availability.

If a number of available bandwidth changes occur up to some time $t_0$ and no other changes occur subsequently, under certain assumptions, all of which are reasonable in a practical network, the convergence of the above algorithm to an optimal solution within a finite time after $t_0$ can be proved, as in Jayant [1992]. These assumptions are as follows:

- Nodes never stop updating their labels and receiving labels from their neighbors.
- All the initial node labels are non-negative. Furthermore, all estimates communicated to nodes by neighbors before the initial time $t_0$ and received after time $t_0$ are non-negative.
- All old label values are eventually purged from the system.

Under these assumptions it can be proved as in Jayant's article [1992] that our algorithm converges to the optimal solution. A key role in the proof is played by the monotonicity property of the dynamic program iteration. In our case, this monotonicity property is expressed as follows.

Given scalars $H_i^1, B_i^1, H_i^2, B_i^2$, which satisfy the inequalities $H_i^1 \le H_i^2, B_i^1 \ge B_i^2$, then

$$min_{j \in N(i)} \{ \alpha [1 + H_j^1] + max\left[\frac{1}{B_j^1}, \frac{1}{b_{ji}}\right] \} \le min_{j \in N(i)} \{ \alpha [1 + H_j^2] + max\left[\frac{1}{B_j^2}, \frac{1}{b_{ji}}\right] \}$$

*This statement applies to all nodes except the source node, whose label values remain fixed.

## 5.5 DISTRIBUTED ASYNCHRONOUS ALGORITHMS FOR MAKING INCREMENTAL CHANGES TO THE MBT TO ACCOMODATE A DYNAMIC ENVIRONMENT

In Subsection 5.3 we described various requested changes that can be made to a multicast session initially set up with a source, a layered stream with its bandwidth characteristics, a network topology, a set of destinations and their bandwidth requests. The algorithms described in that section to accommodate requests 1, 2, and 3 are by their very nature distributed and asynchronous, if there is sufficient residual capacity on the multicast tree to support the new demand.

The case where the available bandwidth on the tree is insufficient for delivery of the requested number of layers to the requesting destination, and case 4 where a new destination requests to be added to the multicast session, can be treated similarly: for purposes of route computation, the requesting destination node in the former case is deleted from the old MBT and treated as in case 4. The algorithms presented in Subsection 5.3 to accommodate case 4 are all modifications of a Dijkstra-like algorithm and use the link-state approach to route computation by assuming that the full network topology and bandwidth availability are known to the processing units that compute the MBT. Here, however, we are interested in distributed asynchronous algorithms.

In the next two subsections, we will describe distributed asynchronous algorithms for adding a new destination to a multicast session. We present two algorithms. The first algorithm follows a "no disruption" approach and finds a path to deliver the largest possible number of stream layers to the new destination without destroying the existing tree structure. The second algorithm finds a path that can deliver the maximum number of layers to the requester even if that path passes through several tree nodes, thereby creating multiple intersecting paths and destroying the tree structure. This algorithm determines paths that destroy the tree structure at a minimal number of nodes, thus causing minimal disruptions to the current multicast destinations. This algorithm is followed by another algorithm to restore the tree structure to the routing paths. These algorithms are modifications of the Bellman-Ford-like algorithm described in Subsection 5.2, in which both the cost function and the labelling rule change to suit our objective. In cases where disruptions occur to the old MBT, an algorithm to restore the tree structure to the routing paths follows the Bellman-Ford-like algorithm.

### 5.5.1 No Disruptions to Old MBT

The dynamic program equations 27, 28, and 29 are run at each node in the network, including the new destination, with the caveat that at a node that belongs to the old MBT, bandwidth label updating is permitted only if the preferred neighbor is the same as before. This caveat prevents disruptions at old MBT nodes and at the same time allows more bandwidth to the new destination, if more bandwidth is available along the old MBT links than was available when the old MBT was computed.

### 5.5.2 Minimal Disruptions to Old MBT

Our objective here is to obtain a maximum-bandwidth path to the new destination, with minimal disruptions to the nodes in the old MBT. As in the cases described in Subsection 5.3, we run the route-finding algorithm in two phases.

- In the first phase, we obtain maximum-bandwidth paths to all the multicast destinations, with minimal disruption to existing nodes in the old MBT. This procedure superimposes a new MBT over the old MBT and may destroy the tree structure at certain nodes.
- In the second phase, an algorithm is run to restore the tree structure to the routing paths.

We describe the two phases as follows.

**First Phase.** The old MBT node labels are retained, and a slightly modified Bellman-Ford-like algorithm is run to obtain a new MBT containing all the multicast destinations. The slight modification lies in the following provision: if we have a choice between two maximum-bandwidth paths from the source to a destination with the same hop count, the path that causes fewer disruptions to the old MBT nodes is chosen. This is accomplished by adding another label, $D_i$, to the set of labels at each node $i$. $D_i$ is the number of disruptions caused to the nodes of the old MBT along the path from the source to $i$. The initial values of $D_i$ are 0 for the source node, and any non-negative integer for all other nodes. Equation 27 is modified as follows:

$$p_i = j \in N(i) \text{ that minimizes } \alpha[1 + H_j] + \beta f(D_j) + max\left[\frac{1}{B_j}, \frac{1}{b_{ji}}\right] \tag{30}$$

where $\beta$ is a small positive constant and *f(.)* is a function that adds 1 to the disruption label of node *j* if connecting *j* to *i* causes a disruption at *i*; otherwise, it returns the disruption label of *j*. Thus,

$$\begin{aligned} f(D_j) &= D_j + 1, \text{if connecting j to i causes a disruption at i} \\ &= D_j \text{ , otherwise} \end{aligned} \tag{31}$$

The bandwidth and hop count labels are updated as before. The node label showing the number of disruptions to old MBT nodes is updated as follows:

$$\begin{aligned} D_i &= f(D_{p_i}) \\ D_1 &= 0 \end{aligned} \tag{32}$$

With the above changes, the dynamic program equations find a path *P* from the source node to node *i*, obtained by connecting preferred neighbors, $p_i$ at node *i*, from *i* all the way to the source node. This connection is such that the cost function $\alpha H(P) + \beta D(P) max_{l \in P}\frac{1}{b_l}$ is minimized, *H(P)* and *D(P)* being the number of hops and the number of disruptions to old MBT nodes on path *P*, respectively. Since $\alpha$ and $\beta$ are chosen to be very small numbers, their contribution to the above cost function can be ignored; thus, the above algorithm finds the maximum-bandwidth path to the new destination. If more than one maximum-bandwidth path is available to the new destination, the path with the smaller value of $\alpha H(P) + \beta D(P)$ is chosen. This smaller value is desirable, since it causes paths with smaller hop counts and with fewer disruptions to old MBT nodes to be chosen. The parameters $\alpha$ and $\beta$ can be chosen so as to reflect the relative importance of choosing paths with small hop counts and paths with minimal disruptions to old MBT nodes. At the conclusion of the above algorithm, each node in the old MBT has two sets of labels: one set corresponds to the old MBT, and the other, new set corresponds to the maximum-bandwidth path to the new destination; while each new node added to the routing paths has just one set corresponding to the maximum bandwidth path to the new destination.

**Second Phase**. The algorithm to restore the tree structure to the routing paths, with minimal disruptions to the old MBT has been presented in Subsection 5.3. This algorithm is distributed and asynchronous and thus can be used in a distributed and asynchronous manner.

## 5.6 SUMMARY

In this section we have presented algorithms for incrementally changing a maximum-bandwidth tree for heterogeneous multicast to accommodate changes in the population of destinations and their traffic demands. The objective is to deliver maximum traffic to each destination, while retaining the tree structure, and to efficiently distribute traffic. Both centralized and distributed asynchronous algorithms were presented. The former algorithms are Dijkstra like, while the latter are Bellman-Ford like, in that both the cost function and the labelling rule change to suit our objective. Two types of algorithms were presented: the first obtains the maximum-bandwidth path to the new destination, subject to the condition that no changes are made to the existing MBT; and the second obtains the maximum-bandwidth path to the new destination, with minimal changes to the existing MBT. For each of the above types of centralized algorithms, we presented two whose computations start from the source and destination nodes, respectively. We have also presented an algorithm to restore the tree structure to the routing paths, if the tree structure of the routing paths is destroyed.

Related issues that merit future investigation are

- Protocols for resource reservation along the maximum-bandwidth paths
- Protocols to update forwarding tables at nodes with each incremental change to the MBT
- Extension of our scenario to include more than one source node.

# 6 CONCLUSION

The new generation of gigabit wide-area networks will require new algorithms and protocols to

- Address unique network characteristics such as high bandwidth-delay products, scarce resources at tandem nodes, and data distortion due to packet loss that outweighs losses due to random bit errors
- Support new services that take advantage of the broadband data paths.

The results obtained in this research project address such issues and include

- Novel methods for recovering lost packets at the receiving end of a wide-area, broadband data path in which packets are discarded because of buffer congestion. These methods include
  - Recovering lost packets by means of parity packets that are inserted into the data stream by the sender
  - Efficient, hardware-based encoder and decoder for generating parity packets and recovering lost packets
  - Intelligent buffer management and interleaving, to mitigate the effect of bursts of discarded packets.
- Techniques for the distribution of real-time traffic streams to multiple receivers; these techniques take into consideration network heterogeneity and variations in the receiver's capabilities and preferences. In particular, we proposed the concept of heterogeneous multicast, which is based on the following techniques:
  - Hierarchical encoding of the streams by the sender: i.e., presenting the source signal by an ordered set of substreams, each representing a different part of the original signal. The ordering of the substreams is such that receiving substream 1 allows the reception of the stream at a relatively low quality; and each other substream enhances the reception quality of the substreams before it.
  - Computation of a distribution tree with the root at the source and a path with maximum capacity leading from the source to each destination. An algorithm was presented for computing such a maximum-bandwidth tree.
  - An efficient algorithm for partitioning the original stream into substreams to optimally match the set of path bandwidths avaliable to the receivers.
  - Algorithms for multicasting a *collection* of real-time streams over a given tree. This technique is a generalization of the situations described above; it introduces new elements of contention among the receivers who wish to receive different subsets that cannot all be supported by the network. We proposed a fair resolution of such contention based on bids placed by each receiver for the streams it wishes to receive, and presented an algorithm for allocating streams to tree links so as to maximize the bid earned by the source.

- Algorithms for maintaining heterogeneous multicast trees to accommodate receivers joining or leaving the session. We presented centralized and distributed algorithms for extending and shrinking the tree, while minimizing the disruption to receivers already connected to the tree.

The importance of the techniques and algorithms developed in this project is their ability to enhance the capabilities of broadband wide-area networks, thereby enabling them to provide improved services. Some of these algorithms have already been implemented and demonstrated on a testbed we constructed under another program.

## REFERENCES

Acampora, A.S. 1989. An overview of lightwave packet networks, IEEE Network, 3(1):29-41 (January).

Ballardie, T., P. Francis, and J. Crowcroft. 1993. "Core Based Trees," presented at ACM SIGCOMM '93, San Francisco, California (September).

Bharath-Kumar, K., and J. Jaffe. 1983. "Routing to multiple destinations in computer networks," *IEEE Transactions on Communications*, Vol. 31, No. 3, pp. 343-351 (March).

Blahut, R.E. 1988. *Theory and Practice of Error Control Codes*, Addison-Wesley, Menlo Park, California.

Burton, H., and D. Sullivan. 1972. "Error and error control," *Proc. of the IEEE*, 60(11):1293-1301 (November).

Chiou, S.N., and V.O.K. Li. 1988. An optimal two-copy routing scheme in a communication network, in *Proc. of INFOCOM'88*, New Orleans, Louisiana.

Clark, G., and J.B. Cain. 1981. *Error Correcting Coding for Digital Communications*, Plenum Press.

Doar, M., and I. Leslie. 1993. "How bad is naive multicast routing?" *Proc. of IEEE INFOCOM'93*, San Francisco, California, pp. 82-89 (April).

Feller, W. 1958. *An Introduction to Probability Theory and Its Applications*, John Wiley & Sons, New York.

Ghanbari, M. 1989. "Two-layer coding of video streams for VBR networks," *IEEE Journal on Selected Areas in Communications*, Vol. 7, No. 5, pp. 771-781 (June).

Gonet, P., P. Adam, and J.P. Coudreus. 1986. "Asynchronous time division switching: the way flexible broadband communications networks," *Proc. Int. Zurich Sem. Digital Communications*, pp. 141-145, Zurich, Switzerland (March).

Hluchyj, H., and M. Karol. 1988. "Queueing in high-performance packet switching," *IEEE Transactions on Communications*, COM-36(12):1587-15976 (December).

Huang, A., and S. Knauer. 1984. "Starlite: a wideband digital switch," *Proc. of Globecom'84*, pp. 121-125.

IEEE. 1989. "Proposed standard: Distributed Queue Dual Bus (DQDB) Metropolitan Area Network," IEEE 802.6 Committee (August).

Jayant, N. 1992. "High quality networking of audio-visual information," submitted for publication (May).

Karlsson, G., and M. Vetterli. 1989. "Packet video and its integration into the network architecture," *IEEE Journal on Selected Areas in Communications*, Vol. 7, No. 5, pp. 739-751 (June).

Maxemchuk, N.F. 1986. "The Manhattan street network," *Proc. of Globecom'86*, pp. 255-261, New Orleans, Louisiana.

Nussbaum, E. 1988. "Communication network need and technologies–a place for photonic switching?" *IEEE Journal on Selected Areas in Communications*, 6(7):1036-1043 (August).

Papadimitriu, C.H., and K. Steiglitz. 1982. *Combinatorial optimization: Algorithms and complexity*, Prentice-Hall, Englewood Cliffs, New Jersey.

Prucnal, P.R., D.J. Blumenthal, and M.A. Santoro. 1987. "12.5 Gbit/sec fiber-optic network using all optical processing," *Electronics Letters*, 23(12):629-630 (June).

Rom, R., and N. Shacham. 1988. "Reconfiguration algorithm for a double-loop token passing ring," *IEEE Transactions on Computers*, 37(2):182-189 (February).

Shacham, N. 1992. "Multipoint communication by hierarchically-encoded data," *Proc. of IEEE Infocom'92*, Florence, Italy, pp. 2107-2114 (May).

Sun, Y., and M. Gerla. 1989. "SFPS: a synchronous fast packet switching architecture for very high speeds," *Proc. of INFOCOM'89*, pp. 641-646, Ottawa, Canada.

Turner, J. 1986. "New directions in communications," *IEEE Communication Magazine*, 24(10).

Waxman, B. 1988. "Routing of Multipoint Connections," *IEEE Journal on Selected Areas in Communications*, Vol. 6, No. 9, pp. 1617-1622 (December).

Waxman, B. 1993. "Performance evaluation of multipoint routing algorithms," *Proc. of IEEE INFOCOM'93*, San Francisco, California, pp. 980-986 (April).

# BIBLIOGRAPHY

Ammar, M., et al. 1993. "Routing Multipoint Connections Using Virtual Paths in an ATM Network," *Proc. of INFOCOM 1993*, pp. 98-105.

Bailly, T. 1980. "A technique for adaptive voice flow in integrated packet networks," *IEEE Transactions on Communications*, 28(3):325-333 (March).

Bertsekas, D., and R. Gallager. 1987. *Data Networks*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Chow, C.-H. 1991. On multicast path finding algorithms, in *Proc. of IEEE INFOCOM'91*, pp. 1274-1283, Miami, Florida (April).

Dalal, Y.K., and R.M. Metcalfe. 1978. "Reverse path forwarding of broadcast packets," *Comm. of ACM*, 21(12):1040-1048 (December).

Deering, S.E., and D.R. Cheriton. 1990. "Multicast routing in datagram internetworks and extended LANs," *ACM Trans. Comp. Systems*, 8(2):85-110 (May).

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

Gopal, I., and J. Jaffe. 1984. "Point to multipoint communication over broadcast links," *IEEE Transactions on Communications*, COM-32(9) (September).

Gopal, A., I. Gopal, and S. Kutten. 1990. "Broadcast in fast networks," in *Proc. of IEEE INFOCOM'90*, pp. 338-347, San Francisco, California (June).

Karol, M., M. Hluchyj, and S. Morgan. 1987. "Input vs. output queueing on a space-division packet switch," *IEEE Transactions on Communications*, COM-35(12):1347-1356 (December).

Kompella, V., et al. 1992. "Multicasting for Multimedia Applications," *Proc. of INFOCOM 1992*, pp. 2078-2085.

Kompella, V., et al. 1993. "Two Distributed Algorithms for Multicasting Multimedia Information," *Proc. of INFOCOM 1993*, pp. 343-349.

Lee, T.T., R. Boorstyn, and E. Arthurs. 1988. "The architecture of a multicast broadband packet switch," *Proc. of IEEE INFOCOM'88*, pp. 1-8, New Orleans, Louisiana (April).

LeGall, D. 1991. "MPEG: A video compression standard for multimedia applications," *Comm. of ACM*, 34(4):47-58 (April).

Shacham, N., and P. McKenney. 1990. "Packet recovery in high-speed networks using coding and buffer management," in *Proc. of IEEE INFOCOM'90*, pp. 124-131, San Francisco, California (June).

Shacham, N., and D. Towsley. 1988. "Resequencing delay and buffer occupancy in selective repeat ARQ with multiple receivers," in *Proc. IEEE INFOCOM'88*, New Orleans, Louisiana.

Schwartz, M. 1987. *Telecommunication Networks*, Addison Wesley, Menlo Park, California.

Tode, H., et al. 1993. "Multicast Routing Algorithm for Nodal Load Balancing," *Proc. of INFOCOM 1993*, pp. 2086-2095.

Towsley, D., and S. Mithal. 1987. "A selective repeat ARQ protocol for a point to multipoint channel," in *Proc. of IEEE INFOCOM'87*, pp. 521-526, San Francisco, California (April).

Turner, J.S. 1988. "Design of a broadcast packet switching network," *IEEE Transactions on Communications*, 36(6):734-743 (June).

Zhang, L., S. Deering, D. Estrin, S, Shenker, and D. Zappala. 1993. "RSVP: A New Resource ReSerVation Protocol," preliminary draft submitted for publication.