Video Multicast in (Large) Local Area Networks

Sergio D. Servetto † Rohit Puri * Jean-Paul Wagner § Pierre Scholtes § Martin Vetterli §*

† School of Electrical and Computer Engineering, Cornell University.
§ Lab. de Communications Audiovisuelles, École Polytechnique Fédérale de Lausanne (EPFL).
* Dept. of Electrical Engineering and Computer Sciences, University of California, Berkeley.

Abstract— **We consider the problem of distributing high-quality video signals over IP multicast in *large* Local Area Networks (LANs), under real-time delay constraints, and with software-only processing. In a large LAN (such as the network of a university campus, or the network of a large company), the source of channel heterogeneity that the video communications system must cope with is not that of different bandwidth constraints available to each receiver, but it is essentially variations in the available CPU power that each receiver will have in order to decode the incoming signal. In this paper we propose a new architecture for a video multicast system, present the design of the different components of this system, and show results obtained in a real implementation. Our feeds consist of video encoded at about 3 Mbits/sec and 16 frames/sec, capable of tolerating the loss of about 300 Kbits/sec worth of data. Decoding is performed on Linux PCs, and the quality of our reconstructed signals degrades gracefully with the speed of the CPU on which the receiver runs.**

I. Introduction

A. The Internet Multicast Backbone (MBone)

In 1990, under the sponsorship of the U.S. Defense Advanced Research Projects Agency, the DARTNet (DARPA Research Testbed Network) was created. The DARTNet was a wide-area network consisting of Unix workstations acting as programmable routers, interconnected via T1 links. Programming these routers with multicast capabilities was easy, and therefore early on a preliminary version of IP multicast was deployed in this network, thus providing a real platform on which to experiment with wide-area, reasonably large scale multicast.

With a multicast infrastructure already in place, it was only a matter of time until applications that made use of it started to appear. By 1991 (about a year after the initial deployment of the DARTNet), *vt*, an audio conferencing application had already become popular, and this sparked interest in extending the multicast infrastructure beyond the DARTNet testbed. Since back then existing Internet routers could not carry multicast traffic, experimental multicast routers were programmed to be able to forward packets and exchange control messages among them over regular point-to-point connections. This was the origin of the MBone, an experimental network of multicast enabled subnets connected via IP tunnels, which started operations in 1992.

The initial success of the MBone motivated further interest in the development of tools that make use of that infrastructure. Examples of audio and/or video conferencing tools are INRIA's *ivs*, Schulzrinne's *nevot* [1], Xerox PARC's *nv* [2], UCB/LBNL's *vic* and *vat* [3], and more. There were also tools that required a different type of multicast service: whereas audio and video data can tolerate some amount of packet loss without severely degrading the quality of the reproduced signal, applications which involve the transport of binary data (such as text) cannot. Examples of tools that require *reliable* multicast (meaning, multicast with some kind of guarantee that all receivers will eventually receive a particular piece of data) are LBNL's *wb* [4] (a shared whiteboard application), and UCL's *nte* [5] (a shared text editor). Most of these tools were already available in 1996.

During its first four years of existence, the MBone went from an initial configuration in 1992 with a couple dozen subnets to one with about 3000 subnets in 1996. In the same period of time, significant advances were made in the design of protocols capable of providing both layered-unreliable and "TCP like"-reliable multicast service on top of the raw best-effort multicast packet forwarding service. Extrapolating from the previous four years, it would have been perfectly reasonable to predict in 1996 that massive deployment of IP multicast in the Internet was only a matter of time, and that by the turn of the century all kinds of multicast applications would be in widespread, commercial use. However, that did not happen.

Contrary to what one would have expected only five years ago, the MBone has not seen a growth until today which is anywhere comparable to that, for example, of the World Wide Web during the same period. There is not a single major TV station broadcasting live over the Internet, and none of the MBone tools gained the popularity that web browsers enjoy today. Furthermore, even though IP multicast did eventually make it into commercial routers, the technology is in general deemed not reliable enough, to the extent that even some research universities turn off completely the multicast capabilities of routers in their internal networks. When compared with the Web over the last five years, it is safe to say that IP multicast has not lived up to expectations. So a natural question arises: why?

B. Shortcomings of Existing Video Multicast Systems

As argued above, applications that make use of IP multicast are very different, and hence it is likely that many factors, not necessarily uniform across applications, have combined to cause the current state of affairs. In the specific context of the application of video distribution we are interested in here, we feel key factors are (a) poor scalability to large numbers of users, (b) poor reliability of existing multicast mechanisms, and (c) poor quality of the signals reconstructed at the receivers.

The state-of-the-art in IP multicast systems for video is based on *layering* techniques: the video source is sliced into a number of multiresolution layers, each layer is transmitted over a separate multicast group, and different receivers adapt to bandwidth fluctuations by adjusting the number of layers to which they subscribe. This is the approach pioneered by McCanne in his thesis [3], and of much of the recent work in this area (e.g., [6], [7], [8], [9]). Now, despite its many successes, layered multicast suffers from three basic deficiencies:

• *Lack of fairness among different multicast sessions.* Without some additional machinery (e.g., RED gateways [10]), it is

in general not possible to guarantee that all receivers in all sessions will subscribe to all the layers their fair share of bandwidth would allow them to [11].

• *Lack of fairness to competing TCP flows*. When a number of multicast flows are present in steady-state in a network, and then a new TCP flow is added, the bandwidth that this TCP flow converges to may be less that one half of its fair share [11], [12].

• *High complexity requirements to scale up to potentially large numbers of users and sessions*. For each multicast session, under existing protocols (RLM-like [11]), the network is required to maintain a significant amount of state information in the routers [3]. And since this amount grows with the number of sessions as well as with the number of receivers in a session, serious concerns are raised about how well these schemes could scale up.

When enough bandwidth is available, from a practical point of view issues with moderate unfairness are not enough to question the use of layered multicast. And furthermore, such fairness issues have been considered in more recent work on reliable multicast at the IETF (see `http://rmt.motlabs.com/`). However, the scalability issue is, since in this case the network may turn out to be unable to provide service in important practical situations (e.g., under the volumes that would be handled by public TV broadcasts).

Besides issues specifically related to *layered* multicast, we believe there are other factors that have contributed to prevent a massive deployment of video services over IP multicast. Salient among these is the unreliability of existing multicast support, which causes severe fluctuations in the throughput and loss rates observed by receivers, translating into fluctuations (i.e., poor quality) of the signals these receivers are able to reconstruct, finally resulting in a lack of demand for these services. This latter point is particularly important: users have high expectations regarding the quality of audio and video services: for most people, "video" means *television quality* video, and "audio" means *telephone quality* audio—that is, high quality signals (as in TV signals) and extremely high reliability (as delivered by the telephone network). So it seems natural that to most people, a small window with video coded at rates in the low hundreds of Kbits/sec and with frequent outages should simply be not appealing enough. Yet this is the best that MBone tools have been able to offer so far.

In summary: we believe that until a video multicast system is deployed, capable of scaling up to very large audiences, and capable of delivering signals of a quality and reliability significantly higher than what current systems deliver today, video over IP multicast will remain at the level of toy applications. The design and implementation of a system that does deal with these issues is the main problem tackled in this paper.

C. The Key Step: Video Multicast in Large LANs

We claim that a key step in the design of a global scale system for the distribution of broadcast-television quality video over IP multicast is the solution of that same problem, but in the simpler context of a *large* LAN (such as, for example, the network of a university campus, or the network of a large company). We observe the following:

• Today, such networks are typically 100 Mbits/sec Ethernets, with native IP multicast capabilities. With such bandwidths, using about 3 Mbits/sec to multicast a good-quality video stream is perfectly feasible.

• On existing, inexpensive PCs, it is perfectly feasible to decode—in real time and only in software—such high quality video encodings. And even though this is not yet comparable to full motion NTSC video (much less to HDTV), such a system would represent a significant improvement in quality, when compared to existing popular unicast streaming solutions.

So, from a pure "technological feasibility" point of view, the design of a system like the one we propose does not pose any problems: in the campus and/or business environments, the *last mile* is already there and with high bandwidth—that is what we take advantage of. Therefore, under the assumption that high-quality video streams can be fed into a LAN, their efficient distribution within the LAN is indeed an important problem.

As for the assumption of being able to feed high-quality video streams into a LAN, we claim that is the simplest of the problems to deal with. This is so because there exist already a number of proven technologies, in widespread use, that can be used to accomplish this task. For example, communication satellites would be a natural candidate technology to bring high-quality video streams all the way to the edge of the network, and do this at a fraction of the cost that would be incurred into by sending everything over IP.[1]) Alternatively (and more conventionally), emerging approaches based on DiffServ could be used to solve this problem as well.

D. Main Contributions and Organization of the Paper

In this paper we present the design, some implementation experiences, and preliminary performance results for a system used to distribute high-quality video over large LANs. Key features of our system are:

• The encoder and decoder run only in software. On a state-of-the-art PC, the encoder produces in real time a bit stream of about 3 Mbits/sec, some amount of FEC protection included.

• Since in the LAN environments we consider bandwidth is not the bottleneck, it is safe to assume that most of the transmitted 3 Mbits/sec will reach all receivers. However, whereas some receivers may run on high-end powerful workstations, others may run in cheap PCs incapable of processing all the incoming data in real-time. Therefore, our decoder needs to be adaptive: the quality of the video signal reproduced scales down gracefully with the power of the CPU on which the receiver runs.

• In one particular LAN environment, we have performed measurements of packet loss statistics. And much to our surprise, we have observed complex dynamics in the packet loss process, that our system needs to deal with if it is going to provide good QoS.

Now, whereas these are all important contributions, we feel that perhaps the single most important contribution of this paper is that of reporting on actual experiences, obtained in a working prototype of our proposed system. A motivation for all of our work came from trying to answer a simple question: what if we didn't have to worry about bandwidth issues in the last

[1]The New York Times published an interesting article on this subject, available from `http://www.nytimes.com/library/tech/99/12/circuits/articles/02next.html`.

mile? Could bandwidth alone solve all the problems of video multicast? Or are there interesting *systems* issues to consider as well? We find that cranking up bandwidth is *not* going to solve all these problems, and we describe a concrete system in which we deal with these issues to some extent.

The rest of this paper is organized as follows. In Section II we present details of our study on the statistics of packet losses in a LAN environment, in Section III we present details on our proposed video coding techniques, and in Section IV we present performance results obtained in a real implementation of the system. The paper concludes with a summary and a discussion of work in progress in Section V.

II. Multicast Packet Loss Statistics

In the design of any communications system, a fundamental problem that needs to be addressed is that of understanding the nature of the noise that the system will have to deal with. For example, in our system: what is the average rate at which packets are lost? Is knowledge of this average enough to design coding strategies that will allow reliable communication even in the presence of losses? Or do we require knowledge of higher order moments of the loss process? And how do we code for this channel?

To start providing answers to these questions, we have performed some measurements of packet losses in a multicast environment. In this section, we first describe the data collection setup, then we show some statistics computed based on those measurements, and we use these statistics to make some simple inferences about the probabilistic structure of the loss process.

A. Data Collection Setup

We performed a number of experiments in two different networks. A first group was collected during the summer of 2001, in the campus network of the Ecole Polytechnique Fédérale de Lausanne (EPFL, Switzerland). Collection of a second group of measurements started in November 2001, in the campus network of Cornell University. The collection of data at Cornell is still in progress—in this paper, we will present results based on the data collected at EPFL.

The experimental setup at EPFL was as follows. We installed a transmitter in `lcavpc36`, and receivers in `icapc3` and `ioapc29` (all Linux PCs within the `.epfl.ch` domain, on three different 100 Mbits/sec Ethernets). The topology of this portion of the network is illustrated in Fig. 1.

The transmitter in `lcavpc36` sends a packet roughly every 4 miliseconds (255 packets/sec), to an IP multicast address (239.255.100.21). The `ttl` field is set to 16, to make sure that these packets do not leave the campus network. At each receiver we record the sequence number and local arrival times for each packet, plus some information common to all packets (IP addresses of transmitter and receiver, time of day, size of the packets, socket buffer size). Initially, 50 traces were collected, each lasting about 1000 seconds, over periods of about 13 hours.

B. Measurements

So as to develop some intuition on what were the patterns we were looking for in this data, we decided to start by looking at some of these traces. A typical trace is shown in Fig. 2.

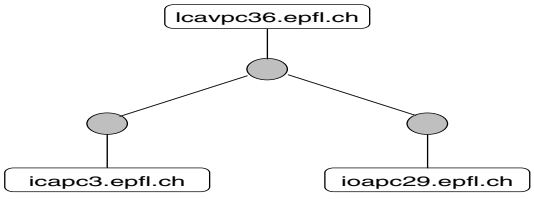

Fig. 1. Topology of the multicast trees under consideration. `lcavpc36` is the transmitter, `icapc3` and `ioapc29` are the receivers, and circles denote internal routers. This configuration was discovered using the `traceroute` and `mtrace` utilities. We have also observed that this tree topology does not change if we move the transmitter from `lcavpc36` to any of the other machines: of course the intermediate routers will change, but we always have a topology with two routers from the transmitter to any receiver—one shared, one private.

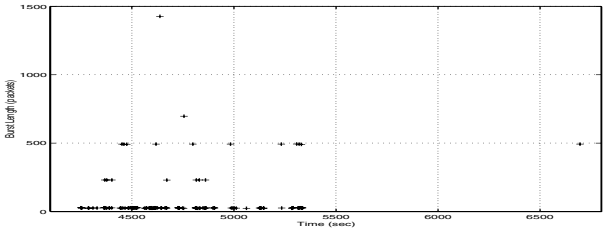

Fig. 2. A section of a raw trace. Observe how there is a period of almost 20 minutes (roughly, between seconds 4300 and 5400) in which there is a large number of bursts of packet losses, most of them short (in the order of 30 packets—at an injection rate of 255 packets/sec, about 1/8-th of a second), but a few long bursts as well. This period of frequent bursts of losses was followed by another period of over 20 minutes (roughly between seconds 5400 and 6700), in which there was not a single lost packet.

The same type of behavior that we observed in the trace of Fig. 2 we observed in many other traces, and lead us to conjecture that the loss process we are dealing with may be in general one in which packets are lost in bursts, and that these bursts in turn tend to cluster over time. To tests these hypotheses we perform two experiments, reported next.

B.1 Bursts of Packet Losses

In our first test, we first process our traces to extract the length of all bursts of packet losses: an isolated loss would count as a burst of length 1, two consecutive losses as a burst of length 2, a loss then a received packet then another loss then another received packet would count as two bursts of length 1, and so on. Then we form a histogram of the length of these bursts. Now consider two possible scenarios:

- If losses were independent, the histogram of burst lengths should obey a geometric law: $\Pr(\text{burst of length } n) = p^n(1-p)$. Plotting then this expression in logarithmic scale as a function of the burst length n, it takes the form $\log(1-p) + n\log(p)$, forming a straight line with slope and zero-crossing determined by p.
- We do know however that long bursts occur with low probability under a geometric model. Therefore, we also consider the possibility that the distribution of burst lengths might have heavy tails (the heavier these tails, the higher the probability assigned to long bursts). For example, if the probability of a burst of length n was dominated by a term of the form $c \cdot n^{-\alpha}$ (for some constants $c, \alpha > 0$), then in logarithmic scale this takes the form $\log(c) - \alpha \cdot \log(n)$.

We see from these considerations that a measure of burstiness of the data is how much the measured log-histogram deviates from a straight line, to resemble a logarithmic curve. In Fig. 3 we plot the log-histogram measured from these traces, and we compare it against a geometric law with parameter estimated from the traces as $\hat{p} = \frac{\text{\# of lost packets}}{\text{total \# of packets sent}}$.

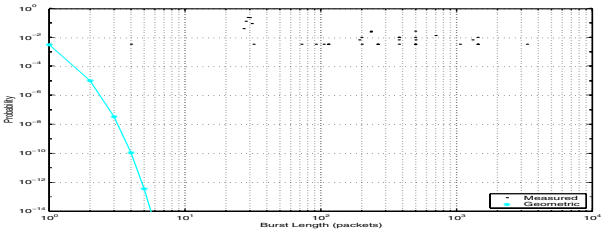

Fig. 3. To illustrate the point that packet losses are highly bursty. Note that since we also plot the horizontal axis in logarithmic scale, the plot of the geometric pmf is not a straight line. The measured log-histogram appears to favor the notion that long bursts of lost packets occur with non-negligible probability.

B.2 Clustered Occurrence of Bursts

The statement that "bursts of losses tend to cluster" means that, over time, there exist periods over which the distance between bursts of losses is small, and there exist other periods during which that distance is large: long bursts of correctly received packets also occur with non-negligible probability. Therefore, we repeat the same test as in the previous subsection, but this time the traces are processed to extract the length of all bursts of *no*-losses. The corresponding plots are shown in Fig. 4.

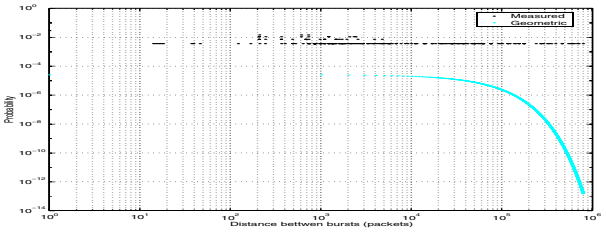

Fig. 4. To illustrate the point that bursts of losses tend to cluster. Here we measure from our traces the log-histogram of runs of correctly received packets, and again, long bursts apparently occur with non-negligible probability.

C. Implications for System Design

In terms of the design of a real-time communications system, these observations are definitely *not* good news. What they say is that, to a first order approximation, the channel appears to behave as if it were switching over time between good and bad states: in the good state there are virtually no packet losses, whereas in the bad state the number of losses can be enough to virtually shut down the channel. The way to deal with this type of impulsive noise is to try to find "diversity" in the system. By diversity we mean mechanisms by which data can be sent over multiple independent "subchannels". The independence requirement among subchannels is crucial, so that even if some of them enter bad states, the chance that all of them do so simultaneously is lower, and therefore the time-critical flow of information is not interrupted.

Unfortunately, with packets going over IP multicast, and with real-time delay constraints, there is not much diversity in the system to exploit. One possibility would be to take advantage of *multipath* diversity: by this we mean sending data over different disjoint paths. But this is not possible, since applications have little/no control over the route followed by each packet. Another possibility is to create the illusion of multiple independent subchannels by means of an *interleaver*: this is a device which produces a random permutation of the data prior to transmission, so that a long burst of losses on the real channel appears like independent losses on the subchannels. But this is not an option either in our case: with bursts which can last for 5-10 seconds, the delay introduced by an interleaver capable of spreading out such long bursts would be unacceptable for our application (in the order of a couple of minutes, at least).

In the current implementation of our system, we have resorted to a rather simplistic approach: we protect the outgoing data stream with a Reed-Solomon code of parameters (255,225,31), capable of correcting up to 30 losses in a block. The choice of parameters for this code is justified in Fig. 5.

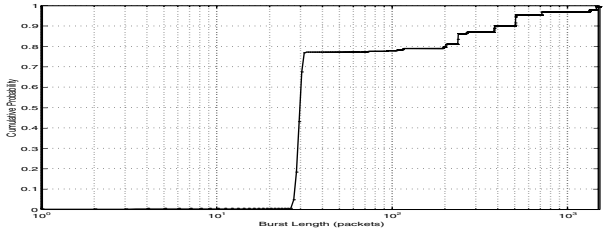

Fig. 5. On the choice of parameters for the Reed-Solomon code. This plot shows the cumulative histogram. We see here that using a code capable of correcting 30 losses we can correct 78% of all the bursts. Combining this with the fact that the relative frequency of lost packets we measured is approx. 0.0032, we have that the probability of not being able to decode one second of video is less than 0.0007—this translates to a mean time between failures of 1420 seconds, or roughly 1 second every 24 minutes.

D. Remarks

We would like to conclude this section with a couple of remarks about the work presented so far.

D.1 On the Irrelevance of Spatially Correlated Losses for Video

There has been work before ours in which the problem of modeling packet loss in multicast had been considered. In studies performed in 1996-97, one by Yajnik, Kurose and Towsley [13], and another by Handley [14], it was observed that there is *both* spatial and temporal correlation in packet losses in the MBone. Although our measurements agree with these previous studies in the suggestion that losses are bursty, there are significant differences worth pointing out:

- Whereas previous work focused on performance of the MBone on a world-wide scale, our work focuses on much smaller network scales.
- An issue studied extensively in previous work is the correlation in losses among nodes close to each other (in the network topology sense). However, that particular kind of structure in the loss process is of limited interest in the context of protocols to support real-time services such as video.

Correlation in the losses seen by nearby nodes has been recognized as an important feature that can be exploited to improve the performance of *reliable* multicast protocols [15]. However, in protocols that support real-time services such as video, the ability to exploit spatial correlations in the loss process is much more limited, since there is often not enough time to complete local repair work. In these cases, data is decoded independently by each receiver, and it is up to the video decoding algorithm to deal with the fact that some information may missing by the time a given video frame needs to be displayed.

D.2 On the Use of Reed-Solomon Codes

The work presented in this section is not a thorough attempt at solving the problem of building a statistical model for the packet loss process in multicast: that modeling task is an interesting problem in itself, and is beyond the scope intended for this paper. Our goal here is to come up with a working system. While working towards that goal, we encountered the problem of long bursts of losses, something we did not anticipate in the context of a LAN. The use of a single Reed-Solomon code is probably not the *best* solution, but it is a *simple* solution chosen because of two reasons: (a) due to its simplicity and low complexity, it allowed us to move forward with our systems work, and (b) while interruptions of one second every 24 minutes are not yet enough to claim TV-like reliability, this is a significant improvement over what MBone tools can claim. Future work will deal with all these important issues, only glanced over here.

III. Complexity-Scalable Software-Only Coding of High-Quality Video

We have seen in Section II that with a simple (and probably suboptimal) form of FEC protection, we can deliver all of the transmitted data to any receiver with high probability: only bursts of length comparable to that of the delay constraints under which our system needs to operate will cause errors, and there is not much that can be done about these. But random losses, and short bursts of losses, are taken care of by means of FEC. Given these observations, as well as the characteristics of the target environment we consider (large, high-speed LANs), in the design of a complete communications system it seems perfectly reasonable to assume that most of the transmitted data will reach its destination.

In most previous work on video multicast, the bandwidth available to each receiver was the key source of heterogeneity that the comm system had to deal with—yet we just saw that such is not the case in our setup. Does this mean our problem is trivial? Not at all. In our case, even if most of the time all the transmitted data makes it to destination, there is yet another source of heterogeneity to deal with: variations in the CPU power of the machines on which the receivers of our system need to run. Recall that our goal is to develop a tool that a large community will find useful and appealing: for that goal we need to come up with a software-only implementation of our receiver, without making any assumptions about the speed of the machine on which these receivers run. Therefore, we see that we have another challenge at hand: that of producing a real-time, software-only, complexity-scalable video coder.

At a coarse level of description, our encoder/decoder pair works as follows. A video signal is split into groups of pictures (GOPs) which are independently encoded, so that decoding of a stream can start at the beginning of any of these GOPs. To encode each GOP, we perform the following steps:

1. Split the GOP into independent components.
2. Quantize and entropy code each component.
3. Protect the encoded stream using FECs.

To decode, we perform the following steps:

1. Decode (FEC) a received GOP.
2. Estimate of how long it will take to decode components.
3. Decide which subset of components maximizes the quality of the reconstructed video signal subject to the constraint that decoding can be completed within a given alloted time—then decode.

A schematic description of the system architecture is shown in Fig. 6.

A. System Components

We see in Fig. 6 that there are three main components that make up our video coding system:

- Split/merge a GOP into independent components.
- Quantization and entropy coding.
- Bit stream syntax.

We provide details next on each one of these components.

A.1 Split/Merge a GOP into Independent Components

At a coarse level, depending on how correlations in the video signal are exploited to achieve good compression performance, video coding algorithms can be grouped into two main camps: coders are either based on *motion compensation*, or on *subband filtering* techniques. Motion compensated coders are known to achieve the highest compression efficiency, although their ability to provide rate/frame/spatial scalability is limited at best. Certain subband coders have been empirically found to not achieve quite the same compression efficiency of motion compensated coders, although they are ideally suited to deal with scalability issues. The discussion on which technique yields the best results is far from settled, and lies outside the intended scope of this paper.

In this work, (a) because of their suitability to deal with the scalability issues of interest to us in this system, (b) because of the excellent systems-related results we obtained, and (c), last but not least, because of our familiarity with the properties of subband data [16], we have chosen to work with a spatio-temporal subband representation of video signals:

- Each subband can be processed (encoded/decoded) independently of every other, without sacrificing compression efficiency.
- Subbands contribute in different (well defined) amounts to the quality of the reconstructed signal, and have different (well defined) requirements in terms of the amount of CPU needed to decode them. This gives rise to interesting complexity/distortion tradeoffs that a smart receiver can take advantage of when pressed for time, by selectively discarding subbands with low contributions to the quality of the reconstructed signals.

The proposed subband decomposition is illustrated in Fig. 7.

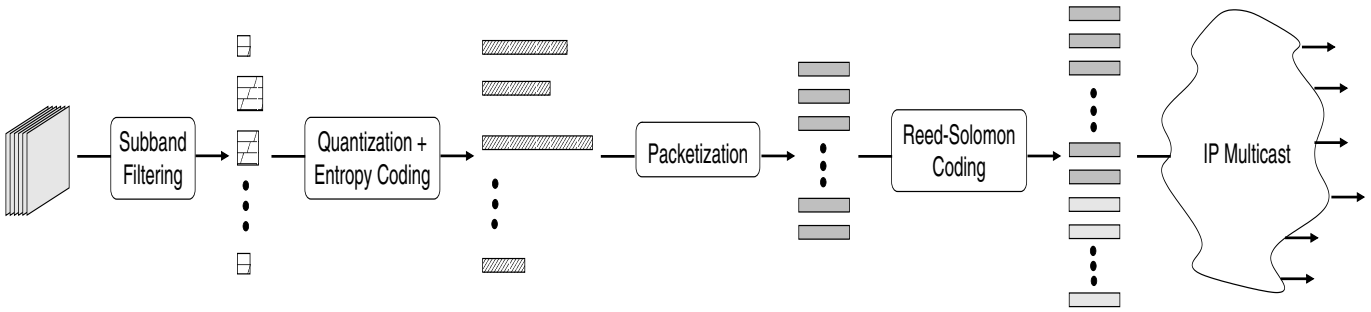

Fig. 6. System architecture. Groups of frames of video are passed through the analysis stage of a filter bank, to produce a set of independent subbands. The coefficients in each of these subbands are quantized, and compressed using Huffman codes, thus producing a variable length bit stream for each subband. All these data bits are byte aligned, and put in packets of equal length. A few parity packets are added, and and then all sent over IP multicast. To decode, the receiver selects a subset of the bit streams to entropy decode and dequantize, performs inverse filtering, and outputs a reconstruction of the original video signal.

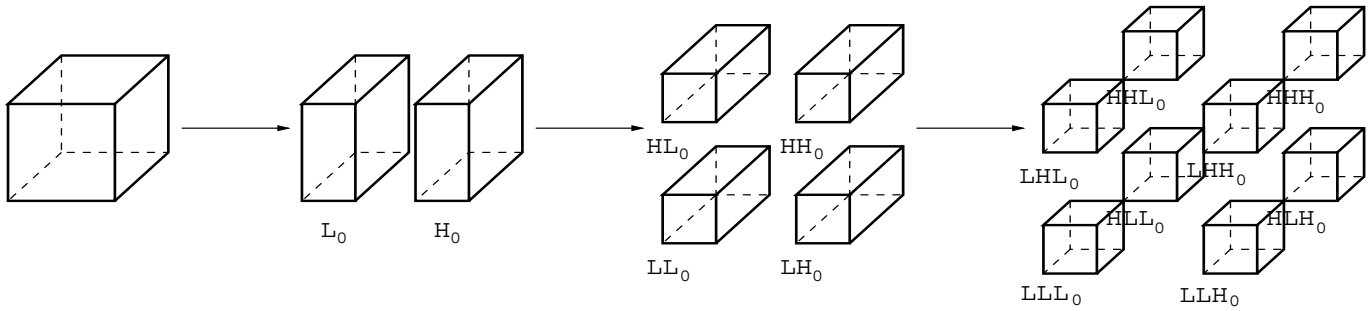

Fig. 7. Splitting a video signal into subband components. A suitable pair of lowpass/highpass filters is applied on a cube of video data of size 16x16x16 (blocks of size 16x16 on each frame, taking 16 consecutive blocks from adjacent frames). By iterating the application of the filter bank on the row, column, and time axes of the cube 8 subbands of size 8x8x8 are produced. Then the process is iterated on subband LLL_0 (the one which results from always applying the lowpass filters in the three axes), to produce another 8 subbands of size 4x4x4, then 2x2x2, then 1x1x1. The same process is applied on all cubes of size 16x16x16 of a GOP and, after processing, all the coefficients in the GOP a that belong to the same subband (LLL_0, HLH_2, etc), are grouped together. These groups of subband coefficients form the independent components into which the video signal is split.

A.2 Quantization and Entropy Coding

Once the original signal is split into subbands, we quantize the subband coefficients using a standard uniform scalar quantizer, and then the quantization indices are compressed using Huffman codes.

Although arithmetic coding is known both from theory as well as from practical experience to perform better than Huffman codes (in terms of delivered compression performance), its computational complexity is significantly higher: an arithmetic coder needs to perform multiple additions and multiplications per encoded symbol, whereas a Huffman encoder/decoder involves simple table lookups.

In practice, when coding data generated by unknown distributions, a key element that explains the high performance of arithmetic coding is its ability to quickly learn these unknown distributions and essentially spend as many bits as if the distribution had been known to start with. In our system, we introduce some a-priori knowledge about these distributions: we model subband data as being generated by some unknown distribution parameterized by the energy level of that subband. Then we use a large number of test video sequences to generate Huffman tables for each subband and for each energy level within a subband. During real-time encoding, the transmitter computes the energy in a given subband (and sends it to the receiver), based on this picks a Huffman table, and uses it to encode all the coefficients in that subband. During real-time decoding, the receiver reads from the bit stream the energy level of each subband, based on this knows which Huffman tables were used to encode each of them, and then decodes. Using this hybrid of training and adaptive rules, we are able to obtain some of the good properties of arithmetic codes without sacrificing the low complexity of Huffman codes.

A.3 Bit Stream Syntax

To protect the compressed data stream against loss of data we use Reed-Solomon codes of length 255, of which 225 are data symbols. The syntax of this data, as well as the way in which data is distributed into packets, is shown in Fig. 8.

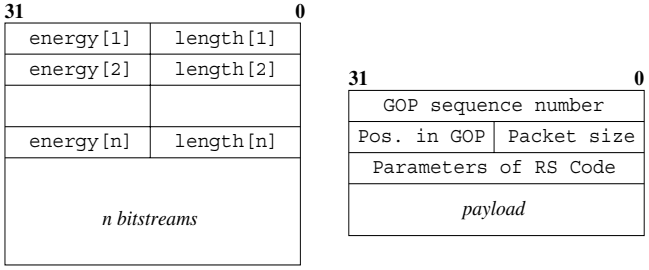

Fig. 8. Bit stream syntax. The *packetization* step in Fig. 6 produces a stream of bytes as with a syntax shown in the left figure. This continuous stream of bytes is then chopped into equal length pieces, parity packets are computed by a Reed-Solomon encoder, which produces packets with a header structure as shown in the right figure.

A.4 Time-Constrained Decoding of Subbands

The algorithm for deciding when to stop decoding subbands is as follows. We assume the decoder does not need to share the CPU with other processes, or that there is some mechanism in place for reserving CPU time—e.g., the real-time extensions to Linux. Then:

- We display 1 GOP per second (16 frames/sec), and we have 29 subbands to decode. Subbands are sorted by energy content: the amount of signal energy contained in any one particular subband is information available to the encoder, and explicitly sent as header bits (see Fig. 8).
- There are two main components of the decoding process that determine the amount of time it takes to decode a subband: entropy decoding and inverse subband filtering:
  - The entropy decoding time is linear in the number of bits to decode. Therefore, we compute an estimate of the time it takes to decode a chunk of n bits (by repeatedly feeding n random bits into the Huffman decoder a fixed number of times and averaging these running times), and then we use it in the rest of the computation.
  - The time to peform inverse subband filtering depends only on the number of coefficients in a subband, and this number is known a-priori. Therefore again, we compute an estimate of the time it takes to decode n subband coefficients (by repeatedly decoding a subband with n values a fixed number of times and averaging these running times), and then we use it in the rest of the computation.
- Based on the above estimates of the time it takes to decode each subband, we reset a timer (set to expire 1 second later), and start decoding them in decreasing order of energy content. When we reach a subband whose estimated decoding time is longer than the time remaining until the timer expires, we stop. If all subbands are decoded before the timer expires, the decoder sits idle, releasing the CPU until the timer does expire, point at which decoding of the next GOP starts.

B. Implementation Details

We have performed a number of optimizations for speed in our code. All our programs are implemented in C (using the `gcc` compiler), for a Linux environment. The routines to compute the subband analysis and synthesis operations were implemented in assembler, to take advantage of MMX instructions. The transmitter has been tested on `lcavpc36`, a Pentium III PC with two 933 Mhz CPUs and 1 Gb of RAM. Receivers have been tested on three platforms:

- One running on `lcavpc36`.
- One running on `lcavpc35`, a Pentium III PC with two 800 Mhz CPUs and 256 Mb of RAM.
- One running on `lcavpc27`, a Pentium III PC with two 450 Mhz CPUs and 256 Mb of RAM.

It is important to mention that our programs do not take advantage of parallel execution threads which can be scheduled on multiple CPUs, so having two CPUs does not make much of a difference. All our programs run as standard user processes—we do not make use of special root primitives, e.g., to raise the priority of some processes, or to pin pages into main memory. We have not yet implemented (although it is in our plans) extensions that make use of the real-time scheduling features of RTLinux. In our initial experiments, we downloaded a live MPEG-2 stream from the european TV satellites ASTRA/HOTBIRD, and we decoded that MPEG-2 stream in a set-top box that produced as output an analog signal. That analog signal was digitized on `lcavpc36` using a Hauppauge WinTV card, compressed using our video coding algorithm, and sent over a multicast socket. At the receiver, data is read from a multicast socket, decoded, and sent both to the X server for display of the images and to the soundcard for audio playback.

IV. Experimental Results

In this section we present results obtained in a real implementation of our system. We classify our experiments into two main groups:

- In one group we explore to what extent the representation of the source induced by our video coding algorithm is suitable for complexity-constrained adaptive processing at the receivers: if the representation is such that without processing all the data generated by the transmitter then the quality of the decoded signals is low, then this is not a useful representation. Our goal here will be to try to develop some criteria for deciding what data to discard and when.
- In the other group, we measure the quality of the signals decoded by the receivers—both in terms of Peak Signal-to-Noise ratios, as well as by means of showing pictures. Of particular interest will be to measure the amount of time required to produce a reconstructed signal of a given quality.

A. Quickly Extracting the Important Data

As in Section II, where the first thing we did was to look at raw trace data in an attempt to develop some intuition on what were the properties of the data that we were interested in, here we will look at the energy distribution across subbands of a couple of typical video signals that our system would have to deal with. Such energy distribution plots are shown in Fig. 9.

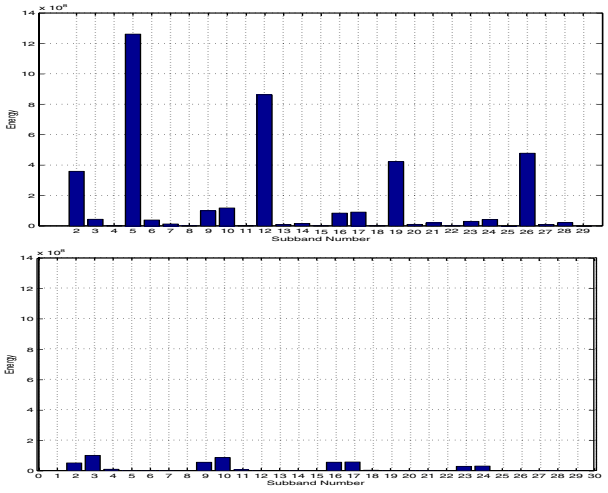

Fig. 9. Energy distribution across subbands. Top: a sequence with high motion content; bottom: a sequence with low motion content. Observe how both sequences have most of their energy concentrated on a few subbands, but also note that these active subbands are not the same. (In these plots, the labels for subbands are arbitrary, but identical in both cases.)

From basic signal processing theory, we know that subband decompositions of a signal (like the one computed in our video coder) tend to concentrate most of the energy in the original signal into a few subbands: that is one of the aspects that make such subband representations useful for compression in the first place. What is interesting to observe in Fig. 9 however is that the *location* of these few active subbands will depend on the content of the signal, thus calling for some form of adaptive processing. Indeed, suppose that due to complexity constraints, the decoder running on a particular machine is only able to decode only a subset of all the transmitted subbands: which subbands should be decoded and which should be discarded?

To illustrate that a one-size-fits-all type of answer is not appropriate here, to show that this decision should depend on the actual signal content, we performed a simple experiment. Suppose that we take a signal with high motion content (we will call these *temporal* signals), and we find what is the best subband to keep if complexity considerations allowed us to decode only one subband, which are the best two if we can afford to decode two subbands, the best three, etc, all the way to decoding all the data. As we increase the number of subbands allowed, we also increase the PSNR of the reconstructed video signal. Suppose now that we use that same ordering for decoding subbands on a *spatial* sequence, i.e., a sequence with low motion content and primarily complexity in the space dimensions. If the ordering made no difference, then the PSNR improvement due to adding subbands should be basically the same with any ordering for decoding subbands. These PSNR profiles are compared in Fig. 10.

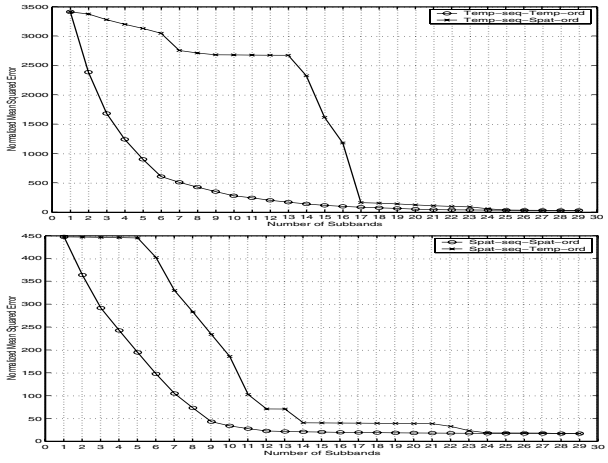

Fig. 10. To illustrate that a careful selection of which subbands to decode can have significant impact in the quality of the reconstructed signal. Observe that in both cases, after decoding 9-10 subbands, the difference in PSNR between the reconstruction obtained by the best ordering of subbands for a given sequence and a mismatched one can be significant: 5-7 dB of PSNR.

At this point there is something we need to emphasize. From a signal processing perspective, the remarks above nearly obvious. But from a systems perspective they are not. What we are saying here is that, in our mechanism for representing a video signal, there are parts of the representation which are more important than others. But unlike in an MPEG stream, where the fact that I frames are more important than P frames and these in turn more important than B frames for the quality of reconstructions does *not* depend on actual video signal being coded, in our representation the importance of the different parts does depend on the content of the signal. Therefore, a receiver capable of looking into the content of packets, and adaptively selecting which ones to decode, in general will be capable of reconstructing signals of a significantly better quality.

B. Computing Good Quality Reconstructions under Time Constraints

The key tradeoff in our application is one of quality against complexity: how much time do we have to spend processing a signal until we can obtain a reconstruction of a given quality? In Fig. 11 we plot the PSNR achievable on the three PCs mentioned above, as a function of the amount of time spent decoding the signal.

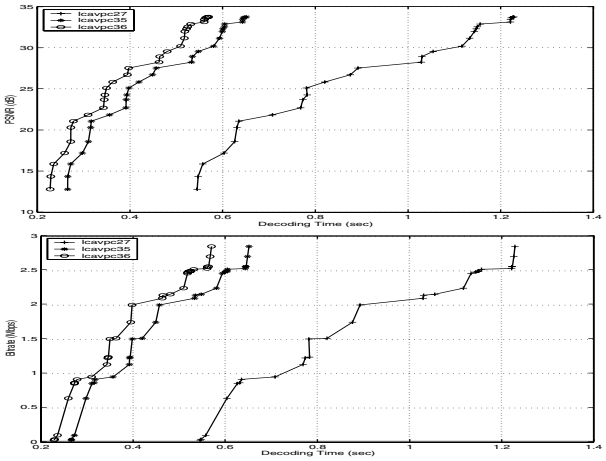

Fig. 11. Complexity-quality tradeoff: these plots essentially show how long it takes for our decoders to process feeds of increasing complexity. Top: PSNR against decoding time; bottom: bit rate against decoding time. "Decoding time" refers to the time it takes to decode a GOP of 16 YUV frames, of size 320x240—it should be possible to decode this much data in 1 second. Observe how the faster PCs can decode the full quality feeds, whereas `lcavpc27` exhausts its time budget after decoding slightly more than 2 Mbits: even so, the PSNR for the GOP is about 27 dB. As a rule of thumb, for video we can say that nearly lossless quality is obtained at about 30 dB PSNR.

To give a more subjective measure of quality, in Fig. 12 we show one video frame decoded by `lcavpc35`, as well as the original (uncoded) frame for comparison.

V. Conclusions

A. Summary

In this paper we have presented our first results in the development of a large scale system for distributing high-quality video feeds over IP multicast. First we argued that solving this problem in the context of a large LAN is indeed an important problem, since feeding high-quality video signals into these environments is nearly trivial (e.g., using existing TV satellites)—the challenge is to maintain that high quality while distributing the signal within the LAN. Then we studied some characteristics of the packet loss process in multicast, and suggested that burst losses may be the type of noise our system needs to deal with. Then we presented the design of complexity-scalable, software-only video encoding and decoding algorithms. We also argued

Fig. 12. Sample reconstructions. Top: an original frame from a GOP with a significant amount of motion (the camera is panning over a relatively complex scene, with rich textures); bottom: a frame from the GOP reconstructed by `lcavpc35` in 0.6 seconds. The PSNR for this GOP is 33 dB—in real-time playback, it is impossible to tell the difference between the original frame and the reconstructed one.

that being able to decode a video signal on any PC (with a quality of reconstructed data proportional to the speed of the underlying CPU) is fundamental to make our system attractive to a large community of users. Finally, we studied the tradeoffs between complexity and performance that are attainable within our framework, and we showed how an existing fast PC (Pentium III 800 Mhz) can decode a full quality 3 Mbits/sec stream, whereas a slower PCs (Pentium III 450 Mhz) can still decode

a reasonable quality signal out of substreams of size roughly 2 Mbits/sec.

B. Distribution over IP Multicast of the Seminar Series on Communications Research at Cornell

Our current work focuses on the implementation of a system for distributing in real time our seminar series on research topics in communications. However, progress on that front has been slower than expected. When we repeated at Cornell the same experiments performed in Switzerland (reported in Section II), we could not make any valid statistical inferences—the data was simply too noisy, we observed traces with *average* (not peak) loss rates of up to 44%, even when injecting less than 2 Mbits/sec; it was on this basis that we decided to not report on these measurements in this paper. This situation we were informed is due to an obsolete and highly error-prone network infrastructure, which is in the process of being replaced as of the writing of this paper. To get a sense for the current state of this network, see Fig. 13.

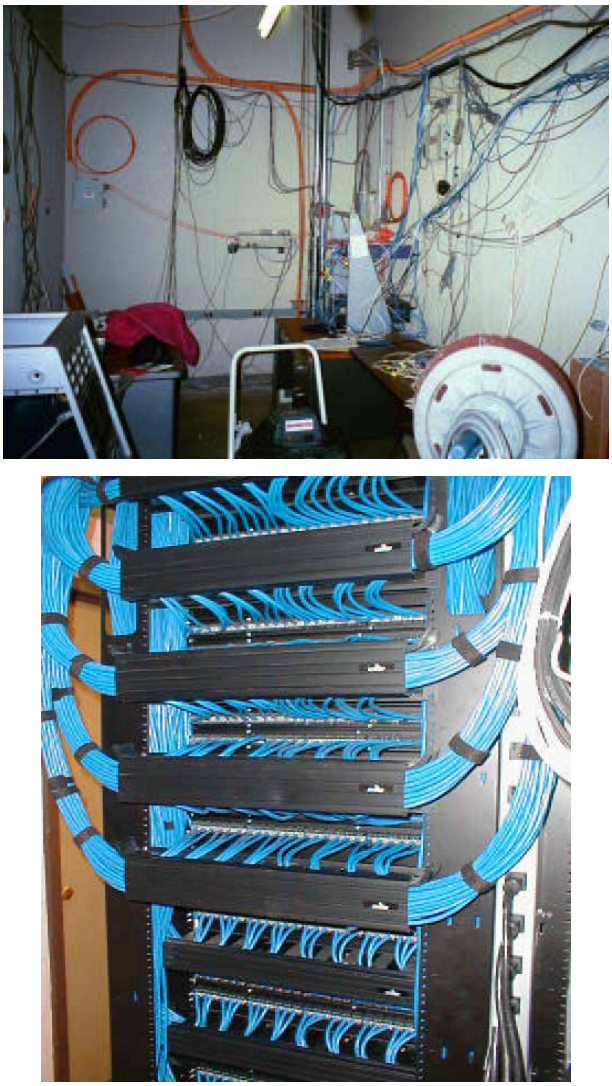

Fig. 13. Two cable rooms, part of the network of the School of Electrical and Computer Engineering at Cornell. Top: one where the old cabling has not been replaced yet; bottom: new infrastructure. The rewiring of our network started around 11/2001, and it is expected that by 4/2002 it will be complete.

As soon as the rewiring process is complete, we expect to be able to start transmitting in real-time our seminars. A web page will be maintained for this project, reachable from `http://people.ece.cornell.edu/servetto/`.

C. Acknowledgements

We would like to thank a number of people that have been of great assistance in the development of our system. The first steps of these work were carried out while the first author was with the Ecole Polytechnique Fédérale de Lausanne (EPFL, Switzerland). At that time, accounts to run some measurements of packet loss in IP multicast were provided by Catherine Boutremans, Thomas Gross, and Michael Unser; Williams Devadason and Richard Timsit also provided assistance with systems questions. At Cornell, Ryan Hsu and Bob Beaver are helping significantly with the set up of an appropriate video lab. The comments made by the anonymous reviewers were particularly useful. Finally, we would also like to acknowledge feedback we received from Professor Don Towsley at an early stage of this work.

REFERENCES

[1] H. Schulzrinne, "Voice Communication across the Internet: A Network Voice Terminal," Tech. Rep. TR 92-50, Univ. of Massachusetts—Amherst, 1992.

[2] R. Frederick, "Experiences with Real-Time Sofware Video Compression," in *Proc. 6th Packet Video Workshop*, Portland, OR, 1994.

[3] S. R. McCanne, *Scalable Compression and Transmission of Internet Multicast Video*, Ph.D. thesis, University of California, Berkeley, 1996.

[4] S. Floyd, V. Jacobson, S. McCanne, and C.-G. Liu L. Zhang, "A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing," in *Proc. ACM SIGCOMM*, Boston, MA, 1995.

[5] M. Handley and J. Crowcroft, "Network Text Editor (NTE): A Scalable Shared Text Editor for the MBone," in *Proc. ACM SIGCOMM*, Cannes, France, 1997.

[6] P. A. Chou, A. E. Mohr, A. Wang, and S. Mehrotra, "Error Control for Receiver-Driven Layered Multicast of Audio and Video," *IEEE Trans. Multimedia*, vol. 3, no. 1, pp. 108–122, 2001.

[7] W.-T. Tan and A. Zakhor, "Video Multicast Using Layered FEC and Scalable Compression," *IEEE Trans. Circ. Syst. Video Tech.*, vol. 11, no. 3, pp. 373–386, 2001.

[8] M. Luby, V. K Goyal, and S. Skaria, "Wave and Equation Based Rate Control: A Massively Scalable Receiver Driven Congestion Control Protocol," IETF Reliable Multicast Transport Working Group Internet-Draft, October 2001, Available on-line at `http://www.ietf.org/internet-drafts/draft-ietf-rmt-bb-webrc-00.txt`. Work in progress. Expires April 2002.

[9] M. Luby, V. K Goyal, S. Skaria, and G. B. Horn, "Wave and Equation Based Rate Control Using Multicast Round Trip Time," Submitted to ACM SIGCOMM, January 2002.

[10] S. Floyd and V. Jacobson, "Random Early Detection for Congestion Avoidance," *IEEE/ACM Trans. Networking*, vol. 1, no. 4, pp. 397–413, 1993.

[11] S. R. McCanne, V. Jacobson, and M. Vetterli, "Receiver-Driven Layered Multicast," in *Proc. ACM SIGCOMM*, Stanford, CA, 1996.

[12] I. Stoica, S. Shenker, and H. Zhang, "Core-Stateless Fair Queueing: a Scalable Architecture to Approximate Fair Bandwidth Allocations in High Speed Networks," in *Proc. ACM SIGCOMM*, 1998.

[13] M. Yajnik, J. Kurose, and D. Towsley, "Packet Loss Correlation in the MBone Multicast Network," in *Proc. IEEE Global Internet Conf.*, London, England, 1996.

[14] M. Handley, "An Examination of MBone Performance," USC/ISI Research Report: ISI/RR-97-450. Available from `http://www.icir.org/mjh/papers.html`.

[15] S. R. McCanne, "Scalable Multimedia Communication: Using IP Multicast and Lightweight Sessions," *IEEE Internet Computing*, pp. 33–45, March/April 1999.

[16] S. D. Servetto, *Compression and Reliable Transmission of Digital Image and Video Signals*, Ph.D. thesis, University of Illinois at Urbana-Champaign, 1999.